# *Analyse et simulation du système Vélib*

## *Analyse et simulation du système Vélib*

**Auteur:Daniel Amaral, Julius Grosskopf,**

**Mateus Orsoni, Zhihao Lyu**

Version:2.0
Statut:Draft
Publication:15/12/2017


**Mines ParisTech**

60 Boulevard Saint-Michel

# Table des matières

# 1 Introduction

Le logiciel permettra d'analyser des données de trajets de Vélib afin de mieux comprendre l'utilisation du système et de proposer et simuler des différents scenarii, pour qu'on puisse améliorer le service. Un fichier CSV contenant les données sur les trajets avec Vélib sera fourni au logiciel, avec ces données on pourra analyser les stations les plus utilisés, où il faut envoyer les camions pour faire la régulation, analyser des nouvelles possibilités.

# 2 Dictionnaire

| Acteur | Description |
|---|---|
| **Analyste** | Ce qui utilise le logiciel. Il définit les paramètres de simulation, demande sa démarrage et regarde les résultat sur le console. |
| **Utilisateur** | C'est l'usager du service Vélib, qui prend et rend les vélos dans les stations. |
| **Régulation** | Retirage de vélo qui a comme objectif équilibrer le nombre de vélo dans les stations. |
| **Station problématique** | Les stations qui restent vide/plein, ou presque vide/plein souvent, les stations auxquelles il faut faire attention. |
| **Paramètres de simulation** | Ensemble des paramètres qui vont définir le comportement de la simulation: le taux de collaboration des utilisateurs, l'augmentation de utilisateurs. |
| **Taux de collaboration** | Paramètre du type *double* qui représente la portion des utilisateurs qui collaborent |
| **Taux de croissance** | Paramètre du type *double* qui représente l'augmentation du nombre d'utilisateurs. |
| **Degré de desequilibre** | Pourcentage de trajets annulés par rapport au total. |
| **Trajet** | un trajet de Vélib effectué dans le monde réel; implémenté par la classe *Trip*; synonyme de *Trip* |
| **Analyste** | Ce qui utilise le logiciel. Il définit les paramètres de simulation, demande sa démarrage et regarde les résultat sur le console. |
| **Utilisateur** | C'est l'usager du service Vélib, qui prend et rend les vélos dans les stations. |
| **Régulation** | Retirage de vélo qui a comme objectif équilibrer le nombre de vélo dans les stations. |
| **Station problématique** | Les stations qui restent vide/plein, ou presque vide/plein souvent, les stations auxquelles il faut faire attention. |
| **Paramètres de simulation** | Ensemble des paramètres qui vont définir le comportement de la simulation: le taux de collaboration des utilisateurs, l'augmentation de utilisateurs. |
| **Taux de collaboration** | Paramètre du type *double* qui représente la portion des utilisateurs qui collaborent |

# 3 Cas d'utilisation

## 3.1 Acteurs

| Acteur | Description |
|---|---|
| **Analyser** | L'analyser est une personne qui utilise le logiciel. Il définit les paramètres de simulation, demande sa démarrage et regarde les résultat sur par la fenêtre. |

*Table 1 Table des acteurs*

## 3.2 Cas d'utilisation

| Cas d'utilisation | Description |
|---|---|
| **LoadData** | L'analyste demande au logiciel à charge des données de base depuis les trois fichiers qu'on possède. |
| **RunData** | L'analyste demande le logiciel à exécuter des trajets générés dans chaque scénario |
| **ShowResults** | L'analyste demande au logiciel montre les résultats par des graphes |
| **ActivateNewBehaviour** | L'analyse active le nouveau comportement |
| **SetGrowthPopularity** | L'analyste fixe un pourcentage d'augmentation de popularité |
| **ShowCollaborationImpact** | L'analyse demande de voir la relation entre la performance du système avec l'augmentation du taux de collaboration |
| **ExportProblematicStations** | Le logiciel exporter la liste des stations problématiques |
| **VisualiseProblematicStations** | L'analyste demande la visualisation des états d'une certaine station. |
| **ShowCancellingPercentage** | Le logiciel montre le pourcentage des trajets annulés |
| **SetConfiguration** | L'analyste fixe des paramètres des scénarios |
| **SetCollaborationRate** | Fixe le taux de collaboration |
| **ShowGrowthImpact** | L'analyse demande de voir la relation entre la performance du système avec l'augmentation de popularité |

*Table 2 Table des cas d'utilisation*

### 3.2.1 Cas d'utilisation "Use Case diagram"

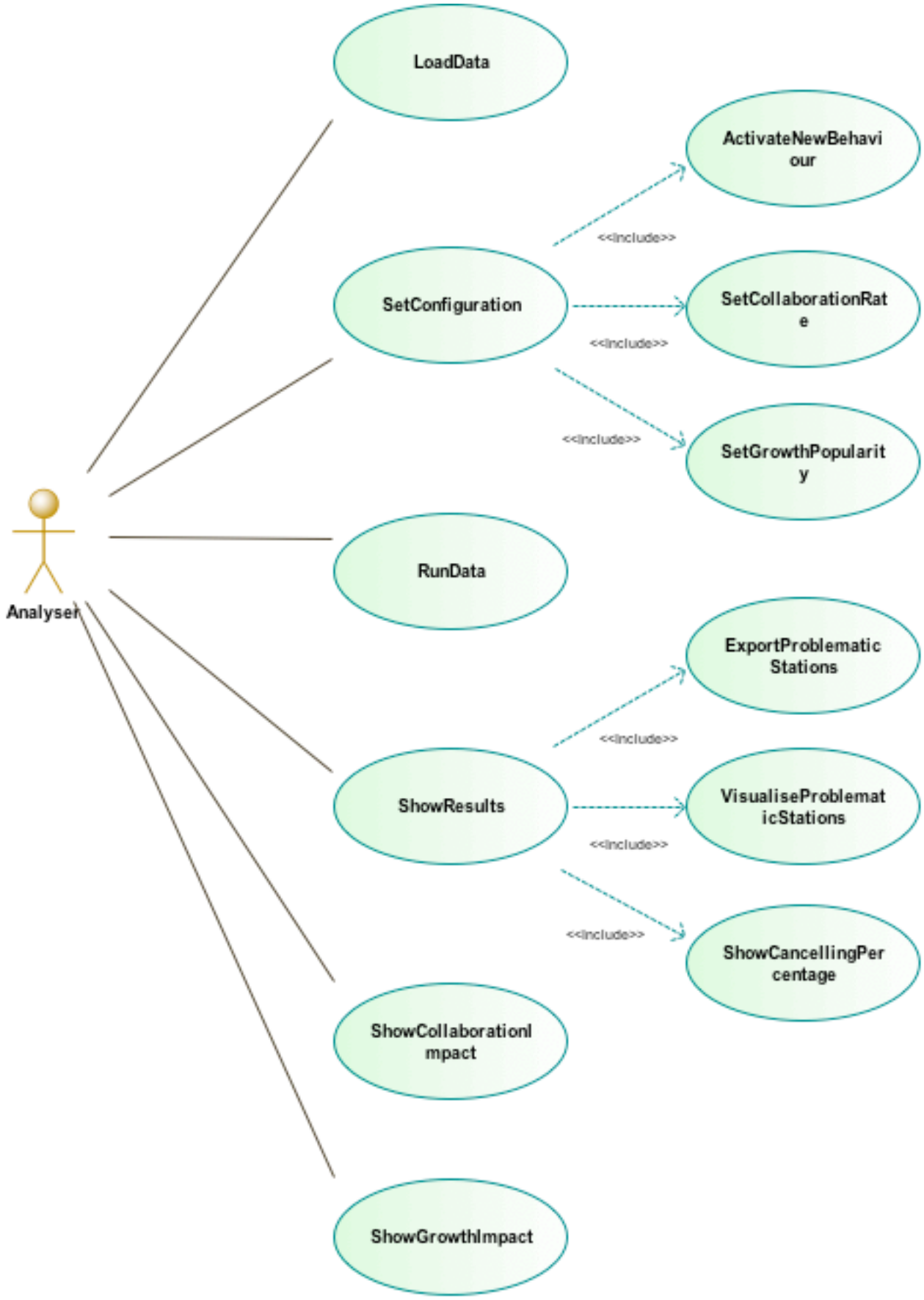


*Figure 1 : Use Case diagram*

# 4 Index des packages

IHM — Ce paquetage est l'interface entre le logiciel et l'utilisateur et charges les données

Data — Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question

Simulation — Ce paquetage contient les classes qui sont en charge de la manipulation des Données pour modéliser le comportement du système Vélib réel

Evaluation — Package pour faire une analyse de la variations des états des stations. Le but est de fournir des visualisations utiles et la possibilité d'exporter des données d'une simulation.

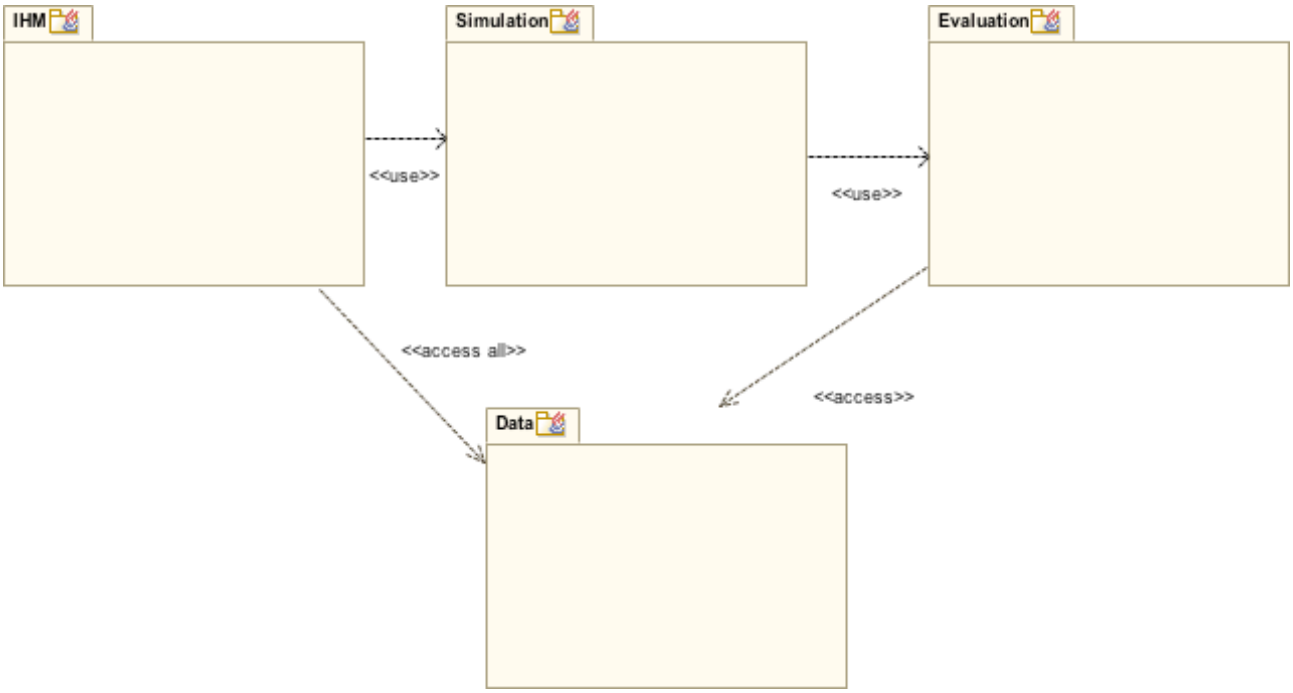


*Figure 2 Class diagram 4.0*

IHM a l'accès total au package Data

| Nom | Résumé |
|---|---|
| **IHM** | Ce paquetage est l'interface entre le logiciel et l'utilisateur et charges les données |
| **Data** | Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question |
| **Simulation** | Ce paquetage contient les classes qui sont en charge de la manipulation des Données pour modéliser le comportement du système Vélib réel |
| **Evaluation** | Package pour faire une analyse de la variations des états des stations. Le but est de fournir des visualisations utiles et la possibilité d'exporter des données d'une simulation. |
| **Documents** | |

*Table 3 Sous packages du package "ProjetVelibDJMZ2"*

# 5 Package "IHM"

*provient de Package* *ProjetVelibDJMZ2*

Stéréotypes : JavaPackage

Ce paquetage contient les classes Read et Window, qui sont responsable d'obtenir les données des fichiers texte et faire la interaction entre le logiciel et l'utilisateur.

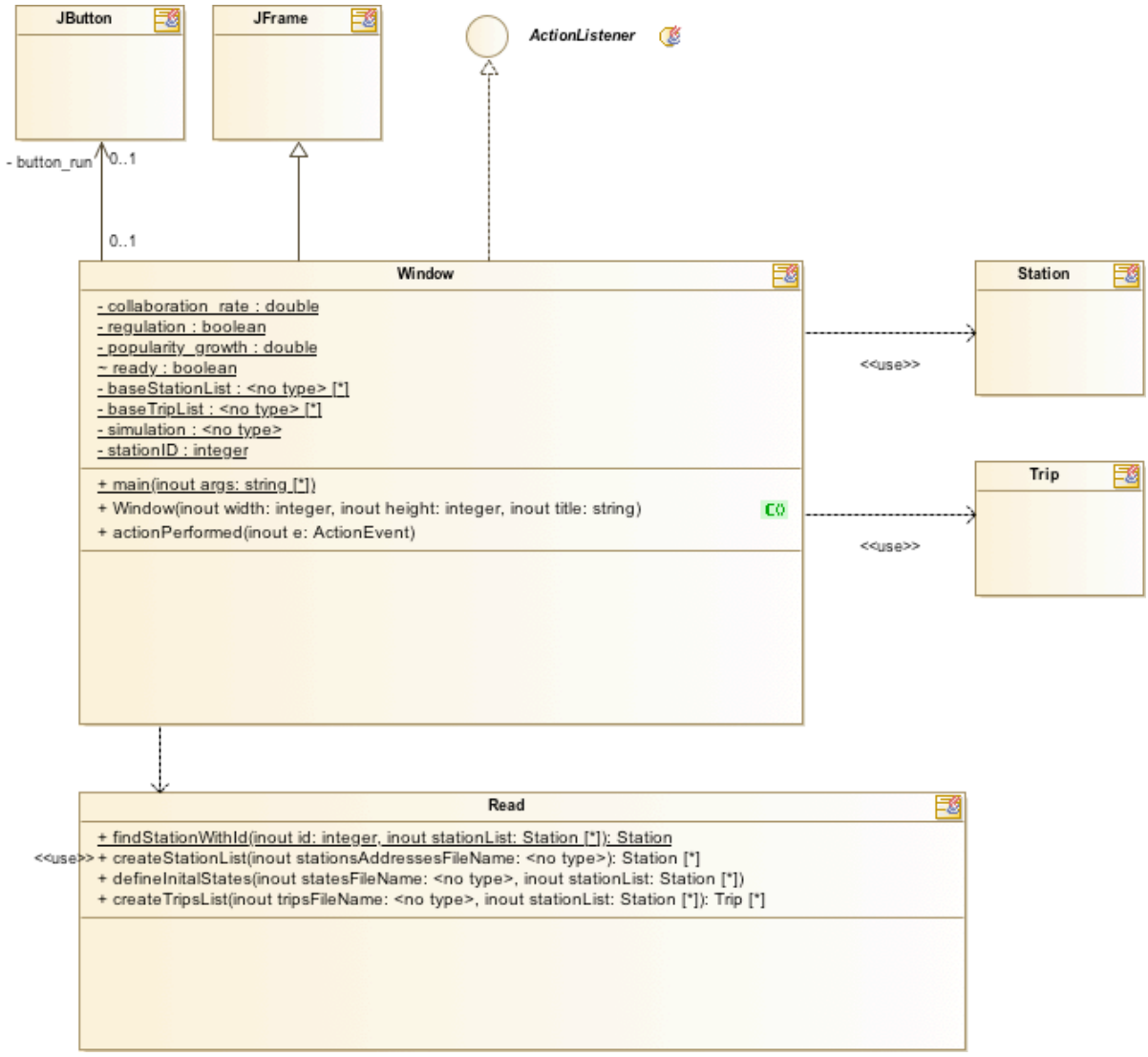


*Figure 3 Class diagram IHM*

| Nom | Résumé |
|---|---|
| Read | La classe Read charge les Données |
| Window | Cette classe implemente une fenêtre pour l'interaction de l'utilisateur avec le logiciel |

*Table 4 Classes appartenant au package "IHM"*

| Nom | Résumé |
|---|---|
| Data | : Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question |
| Data | : Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question |
| Data | : Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question |

*Table 5 Eléments importés par le package "IHM"*

## 5.1 Classe "Read"

*provient de Package ProjetVelibDJMZ2.IHM*

Stéréotypes: JavaClass

La classe Read a trois méthodes principales et une méthode auxiliaire qui font la lecture des fichiers stationsAddresses.txt, initialstates.txt et trips-2013-10-31.txt pour injecter les données dans le systeme. La méthode createStationList prend comme argument un String contenant l'adresse "src/files/stationsAddresses.txt" et retourne le catalogue de Stations comme un ArrayList <Stations>. La méthode defineInitalStates prend comme arguments un String contenant l'adresse "src/files/initialstates.txt" et la liste de Stations créée; les Stations sont trouvées dans la list à partir de la méthode findStationWithId, prenant comme arguments le initier ID de la station e la list elle même, et chaque état initial contenu dans le fichiers sera associé à la bonne Station à partir d'un simple setInitialState. Après ces deux méthodes, le système aura une list de tous les stations dûment initialisées, qui seront utilisées exhaustivement dans le logiciel. La méthode createTripsList prend comme argument un String "src/files/trips-2013-10-31.txt" et la list de Stations et lit son contenu pour créer une ArrayList contenant tous les Trips du fichier. Notez que chaque fichier a une organisation différente des données: les fichiers stationsAddresses.txt et initialstates.txt sont organisés en une seule ligne, pendant que trips-2013-10-31.txt est organisé en une ligne pour chaque trip.

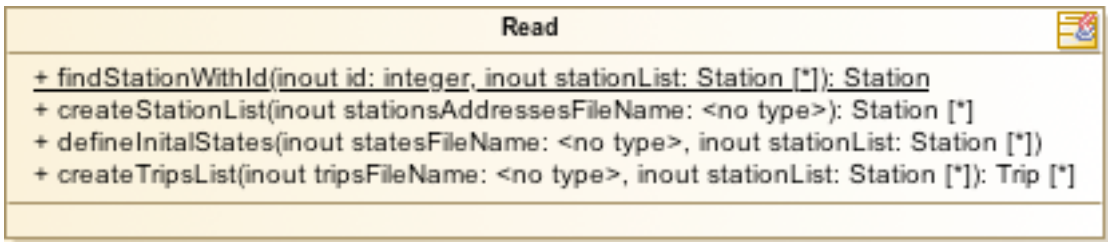


*Figure 4 Read Class diagram*

| Nom | Description |
|---|---|
| **Station findStationWithId (Inout id integer,Inout stationList Station)** | : Prend un ID (int) et la ArrayList et trouve la Station qui a cet ID |
| **Station createStationList (Inout stationsAddressesFileName )** | Prend le String "src/files/stationsAddresses.txt" et retourne l'ArrayList de tous les Stations |
| **defineInitalStates (Inout statesFileName ,Inout stationList Station)** | Prend le String "src/files/initialstates.txt" et la liste de Stations pour "setter" les conditions initiales de chaque Station dans la liste |
| **Trip createTripsList (Inout tripsFileName ,Inout stationList Station)** | Prend le String "src/files/trips-2013-10-31.txt" et la liste de Stations et retourne un ArrayList de tous les Trips du 31 octobre 2013 |

*Table 6 Opérations de la classe "Read"*

## 5.2 Classe "Window"

*provient de Package ProjetVelibDJMZ2.IHM*

Implémente: ActionListener

Hérite de: JFrame

Stéréotypes: JavaClass

La classe Window implémente la fenêtre à partir laquelle l'utilisateur peut interagir avec le système, en héritant la classe JFrame et implémentant l'interface ActionListener pour pouvoir créer l'interface graphique et reconnaître les entrées, et fait aussi le rôle de fonction main du logiciel. La fenêtre possède 4 boutons : Run, qui initialise la simulation, et Show, Population growth et Collaboration rate pour montrer des différents résultats de chaque analyse, selon le cahier des charges. La classe contient plusieurs attributs représentant chaque objet visuel de la fenêtre (buttons, boite de texte et labels) et aussi des attributs statiques utilisées dans la fonction main.

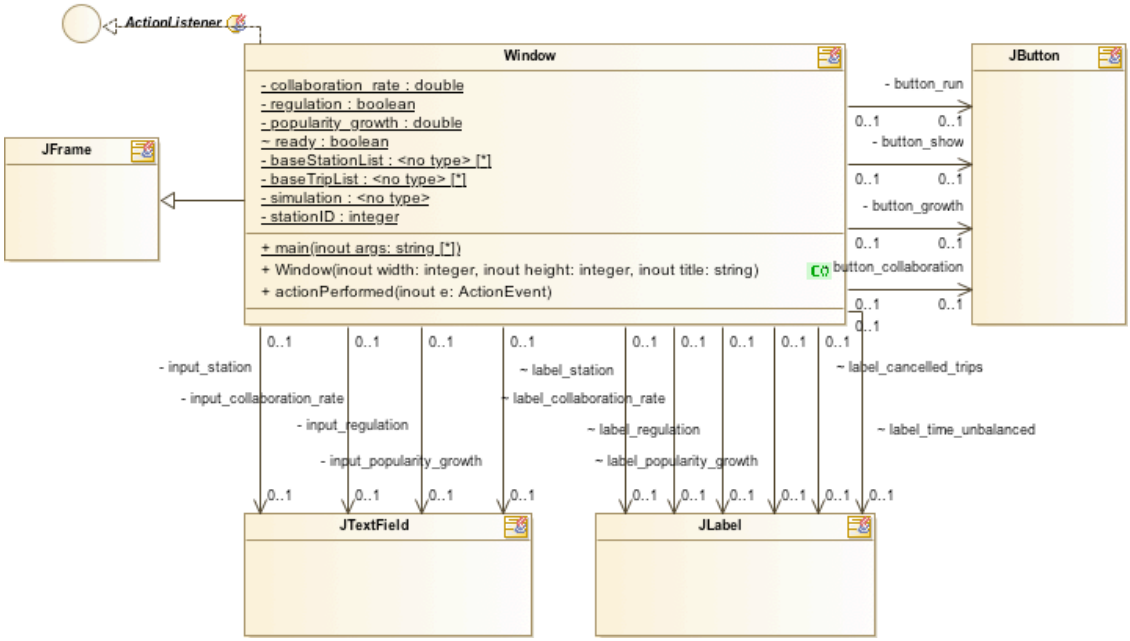


*Figure 5 Window Class diagram*

| Nom | Description |
|---|---|
| **main (Inout args string)** | Fonction main principal du logiciel, où sont chargées les données à partir de la classe Read |
| **Window (Inout width integer,Inout height integer,Inout title string)** | Constructeur de la classe Window, prenant la hauteur, la largeur et le titre de la fenêtre et mettant en place tous les objets visuels |
| **actionPerformed (Inout e ActionEvent)** | reconnaisse une entrée venue de l'utilisateur |

*Table 7 Opérations de la classe "Window"*

| Nom | Description |
|---|---|
| **collaboration_rate : [1..1] double** | Le taux de collaboration fourni par l'utilisateur |
| **regulation : [1..1] boolean** | Regulation considérée ou non pas considérée, fourni par l'utilisateur |
| **popularity_growth : [1..1] double** | Le taux croissance de la population, fourni par l'utilisateur |
| **ready : [1..1] boolean** | variable auxiliaire pour indiquer que le système est prêt pour recevoir des entrées |
| **baseStationList : [0..*]** | Liste de Stations |
| **baseTripList : [0..*]** | Liste de Trips |
| **simulation : [1..1]** | Variable qui garde la dernière simulation faite |
| **stationID : [1..1] integer** | ID de la Station fournie par l'utilisateur |

*Table 8 Attributs de la classe "Window"*

| Nom | Description |
|---|---|
| **->button_run : [0..1] JButton** | bouton de la fenêtre |
| **->button_show : [0..1] JButton** | bouton de la fenêtre |
| **->button_growth : [0..1] JButton** | bouton de la fenêtre |
| **->button_collaboration : [0..1] JButton** | bouton de la fenêtre |
| **->input_collaboration_rate : [0..1] JTextField** | boite d'entrée de texte de la fenêtre |
| **->input_regulation : [0..1] JTextField** | boite d'entrée de texte de la fenêtre |
| **->input_popularity_growth : [0..1] JTextField** | boite d'entrée de texte de la fenêtre |
| **->input_station : [0..1] JTextField** | boite d'entrée de texte de la fenêtre |
| **->label_collaboration_rate : [0..1] JLabel** | label de texte de la fenêtre |
| **->label_regulation : [0..1] JLabel** | label de texte de la fenêtre |
| **->label_popularity_growth : [0..1] JLabel** | label de texte de la fenêtre |
| **->label_station : [0..1] JLabel** | label de texte de la fenêtre |
| **->label_cancelled_trips : [0..1] JLabel** | label de texte de la fenêtre |
| **->label_time_unbalanced : [0..1] JLabel** | label de texte de la fenêtre |

*Table 9 Associations de la classe "Window"*

# 6 Package "Data"

*provient de Package* *ProjetVelibDJMZ2*

Stéréotypes : JavaPackage

Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question

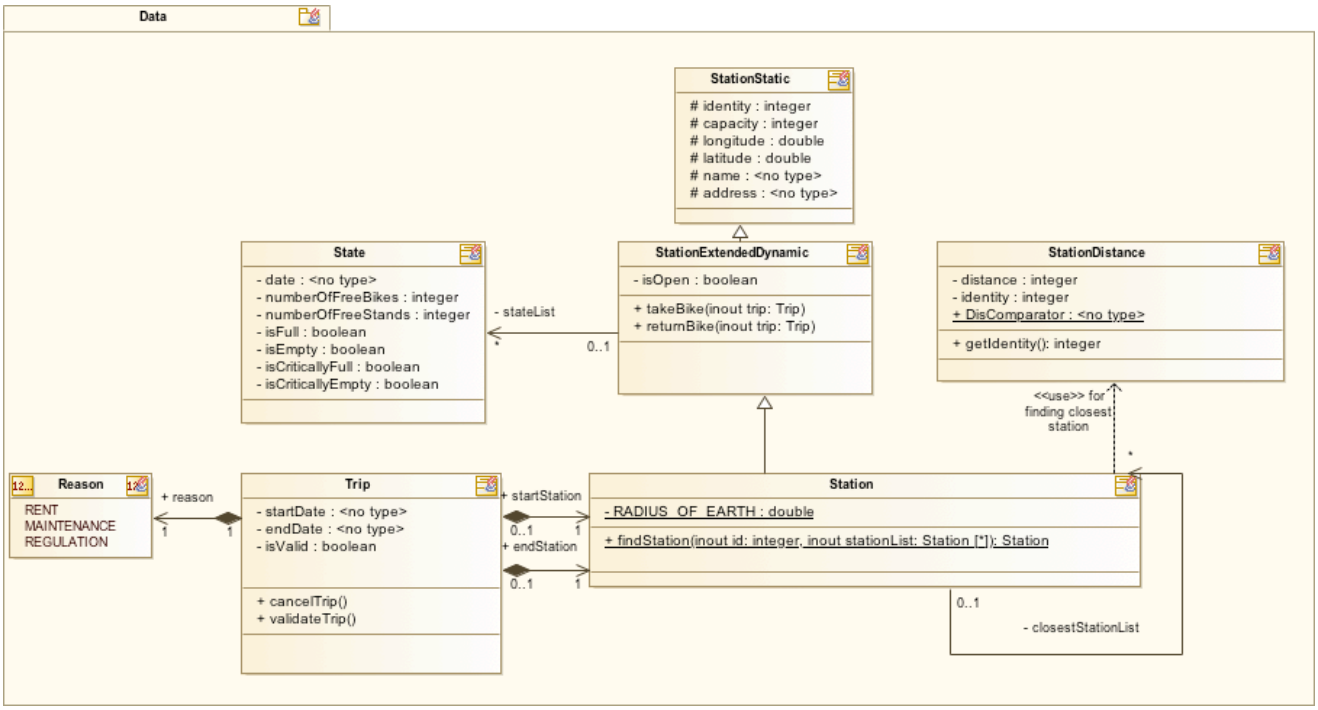


*Figure 6 Data Class diagram*

Ce diagramme montre les associations parmi les classes et des attributs et des méthodes essentiels pour chaque package. Il y a une hiérachie de Station qui s'évolue des informations statiques (identity, capacity, longitude, latitude, name, address)jusqu'aux informations complètes contenant des plus proches stations, à travers de l'ajout des états des station.

| Nom | Résumé |
|---|---|
| **Station** | Classe fille de StationExtendedDynamic, contenant la list des plus proches stations et des méthodes correspondantes |
| **Trip** | Classe de trajet contenant les dates et stations propre à chaque trajet |
| **StationDistance** | Classe utilisé uniquement par Station pour obtenir les identité des stations plus proches |
| **State** | Classe décrivant l'état d'une station, notamment nombre de vélos |
| **StationExtendedDynamic** | Classe mère de Station et classe fille de StationStatic contenant des données dynamique et des méthodes correspondantes |
| **StationStatic** | Classe mère de StationExtendedDynamic contenant des données statiques et des méthodes correspondantes |

*Table 10 Classes appartenant au package "Data"*

| Nom | Valeurs | Description |
|---|---|---|
| **Reason** | RENT<br>MAINTENANCE<br>REGULATION | |

*Table 11 Enumerations appartenant au package "Data"*

## 6.1 Classe "Station"

*provient de Package ProjetVelibDJMZ2.Data*

Hérite de: StationExtendedDynamic

Stéréotypes: JavaClass

Pour chaque station existante, on instancie une classe Station qui contient trois type des informations. Premièrement, les données statiques, comme identifiant, nom, adresse, capacité, longitude, latitude. Deuxièmement, les données dynamiques, des états de stations (le nombre de vélo libre et le nombre d'embarquement libre), et l'ouverture de station. Troisièmement, des informations supplémentaires, notamment des stations les plus proches. À part des données, la classe Station possède aussi des méthodes destinées à traiter la prise et retourne des vélos, et plusieurs méthodes pour trouver des stations les plus proches.

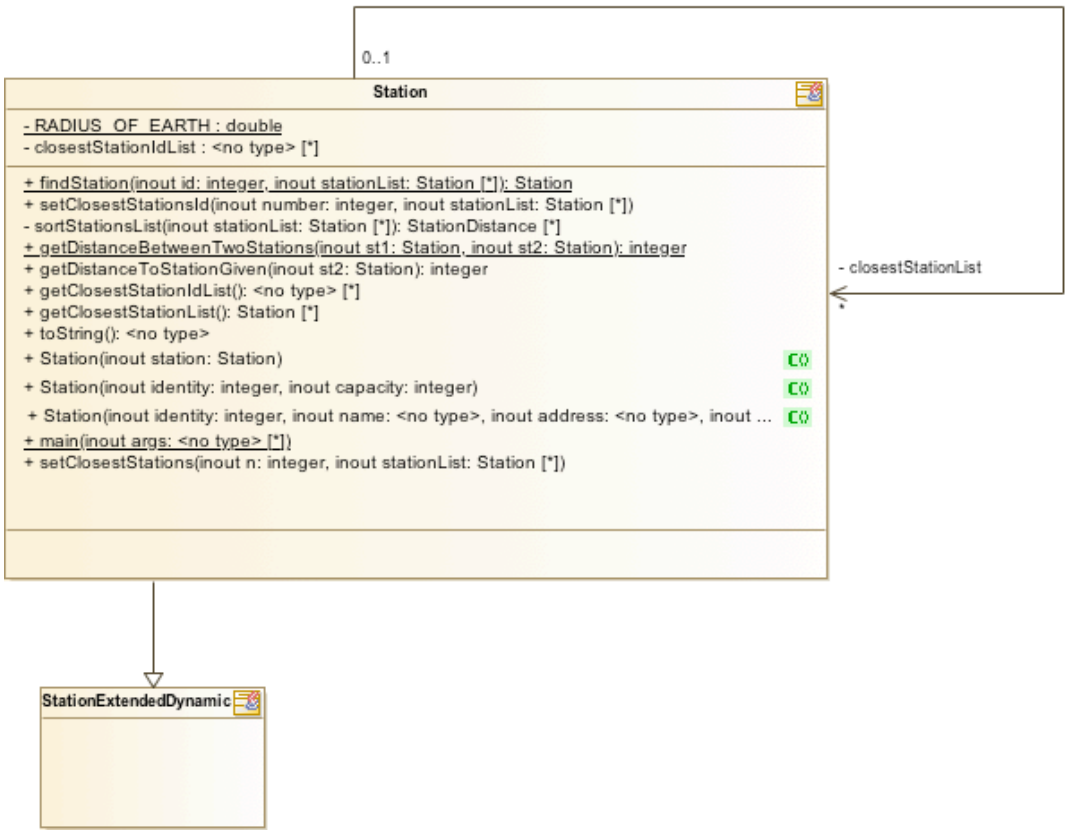


*Figure 7 Station Class diagram*

Diagramme détaillé de la classe Station

| Nom | Description |
|---|---|
| **Station (Inout identity integer,Inout capacity integer)** | Constructeur des informations préliminaires |
| **Station (Inout identity integer,Inout name ,Inout address ,Inout capacity integer,Inout longitude double,Inout latitude double)** | Constructeur des informations complètes |
| **Station findStation (Inout id integer,Inout stationList Station)** | Opération statique pour trouver la station correspondante à une identité |
| **setClosestStationsId (Inout number integer,Inout stationList Station)** | Opération de nature «set» prenant une ArrayList des stations et extraitant l'identité des stations plus proches |
| **StationDistance sortStationsList (Inout stationList Station)** | Opération privée utilisé uniquement par setClosestStationsId |
| **integer getDistanceBetweenTwoStations (Inout st1 Station,Inout st2 Station)** | Opération calculant la distance entre deux stations |
| **integer getDistanceToStationGiven (Inout st2 Station)** | Opération calculant la distance entre «this» station et une station donné |
| **getClosestStationIdList ()** | Opération de nature «get» |
| **Station getClosestStationList ()** | Opération de nature «get» |
| **toString ()** | Opération servant à l'affichage |
| **main (Inout args )** | Opération de test |
| **Station (Inout station Station)** | Constructeur de copying |
| **setClosestStations (Inout n integer,Inout stationList Station)** | Opération de nature «set» prenant une ArrayList des stations et extrayant les stations plus proches elles-mêmes |

*Table 12 Opérations de la classe "Station"*

| Nom | Description |
|---|---|
| **closestStationIdList : [0..*]** | ArrayList des plus proches stations avec la distance croissante |
| **RADIUS_OF_EARTH : [1..1] double** | Constant pour calculer la distance entre deux stations à partir de la longitude et de la latitude |

*Table 13 Attributs de la classe "Station"*

| Nom | Description |
|---|---|
| **->closestStationList : [0..*] Station** | ArrayList des plus proches stations avec la distance croissante |

*Table 14 Associations de la classe "Station"*

## 6.2 Classe "Trip"

*provient de Package ProjetVelibDJMZ2.Data*

Stéréotypes: JavaClass

Chaque trip contient deux instances de Date et deux instances de Station, correspondant respectivement aux départ et terminaison du trajet. Vu qu'on va générer des nouveaux trajets basé sur les trajets qu'on possède déjà, il y a aussi dedans des méthodes pour « set » ces deux instances de Date et deux instances de Station.

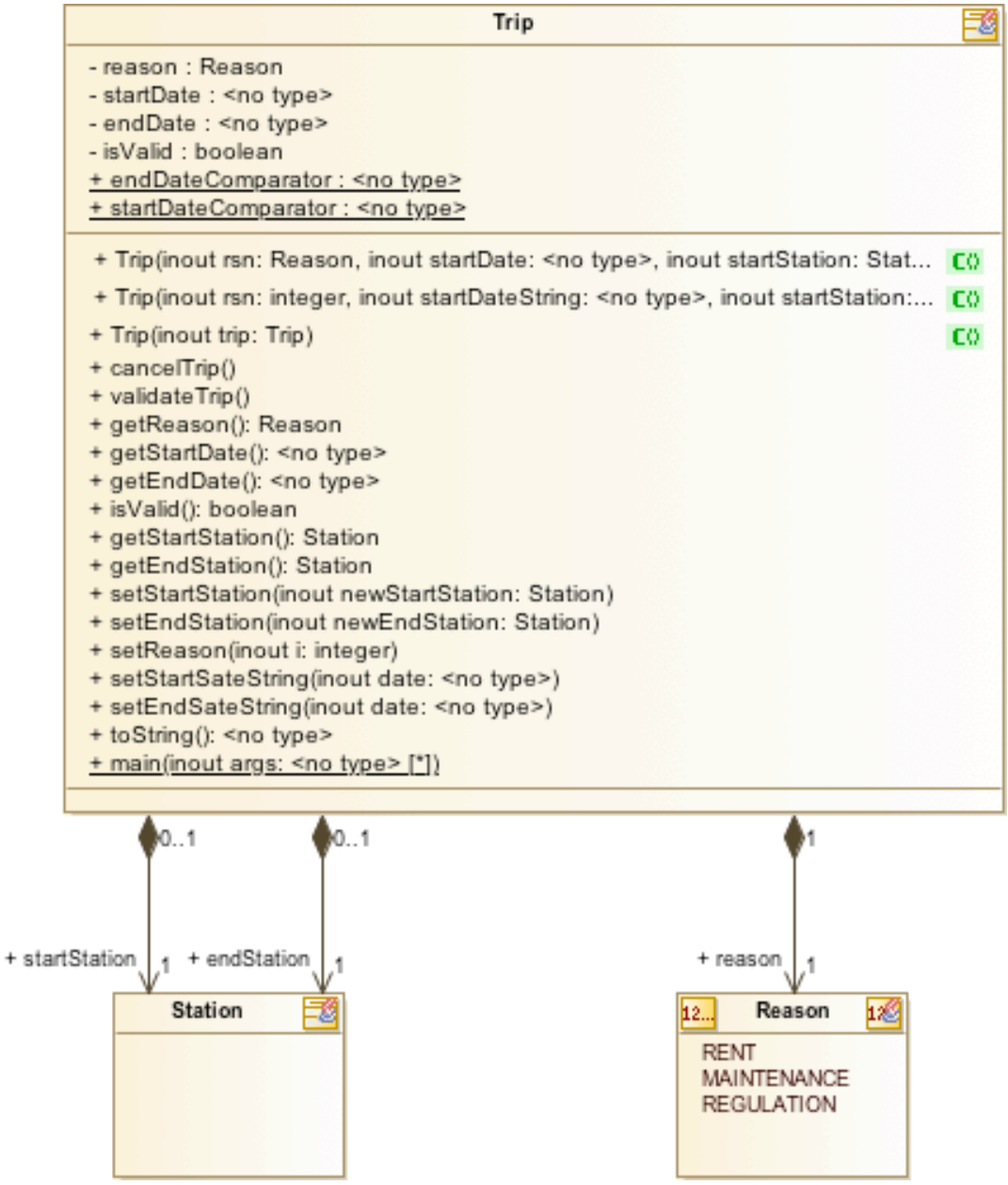


*Figure 8 Trip Class diagram*

Diagramme détaillé de la classe Trip

| Nom | Description |
|---|---|
| **Trip (Inout rsn Reason,Inout startDate ,Inout startStation Station,Inout endDate ,Inout endStation Station)** | Constructeur prenant la raison en forme de Reason et la Date en forme de classe «Date» |
| **Trip (Inout rsn integer,Inout startDateString ,Inout startStation Station,Inout endDateString ,Inout endStation Station)** | Constructeur prenant la raison en forme de Integer et la Date en forme de String |
| **Trip (Inout trip Trip)** | Constructeur de copying |
| **cancelTrip ()** | Opération annulant le trajet |
| **validateTrip ()** | Opération validant le trajet |
| **Reason getReason ()** | Opération de nature «get» |
| **getStartDate ()** | Opération de nature «get» |
| **getEndDate ()** | Opération de nature «get» |
| **boolean isValid ()** | Opération de nature «get» |
| **Station getStartStation ()** | Opération de nature «get» |
| **Station getEndStation ()** | Opération de nature «get» |
| **setStartStation (Inout newStartStation Station)** | Opération modifiant le trajet servant à générer de nouveaux trajets |
| **setEndStation (Inout newEndStation Station)** | Opération modifiant le trajet servant à générer de nouveaux trajets |
| **setReason (Inout i integer)** | Opération modifiant le trajet servant à générer de nouveaux trajets |
| **setStartSateString (Inout date )** | Opération modifiant le trajet servant à générer de nouveaux trajets |
| **setEndSateString (Inout date )** | Opération modifiant le trajet servant à générer de nouveaux trajets |
| **toString ()** | Opération servant à l'affichage |
| **main (Inout args )** | Opération de test |

*Table 15 Opérations de la classe "Trip"*

| Nom | Description |
|---|---|
| **reason : [1..1] Reason** | Attribut de la raison |
| **startDate : [1..1]** | Attribut du début du trajet |
| **endDate : [1..1]** | Attribut de la terminaison du trajet |
| **isValid : [1..1] boolean** | Attribut boolean signifiant si le trajet est exécuté avec succès |
| **endDateComparator : [1..1]** | Comparator comparant la date de la terminatison du trajet |
| **startDateComparator : [1..1]** | Comparator comparant la date du début du trajet |

*Table 16 Attributs de la classe "Trip"*

| Nom | Description |
|---|---|
| **->startStation : [1..1] Station** | Un trajet contient une station de début |
| **->endStation : [1..1] Station** | Un trajet contient une station de terminaison |
| **->reason : [1..1] Reason** | Un trajet contient une raison |

*Table 17 Associations de la classe "Trip"*

## 6.3 Classe "StationDistance"

*provient de Package ProjetVelibDJMZ2.Data*

Stéréotypes: JavaClass

Cette classe est auxiliaire à la classe Station pour ne pas ajouter trop de choses dans la station qui traite le problème de la recherche des plus proches stations. Chaque StationDistance possède deux informations : distance et identifiant, et un « comparator » de Distance. Cette distance a été utilisé pour stocker les distances entre deux stations (une station de repère et une station repérée).

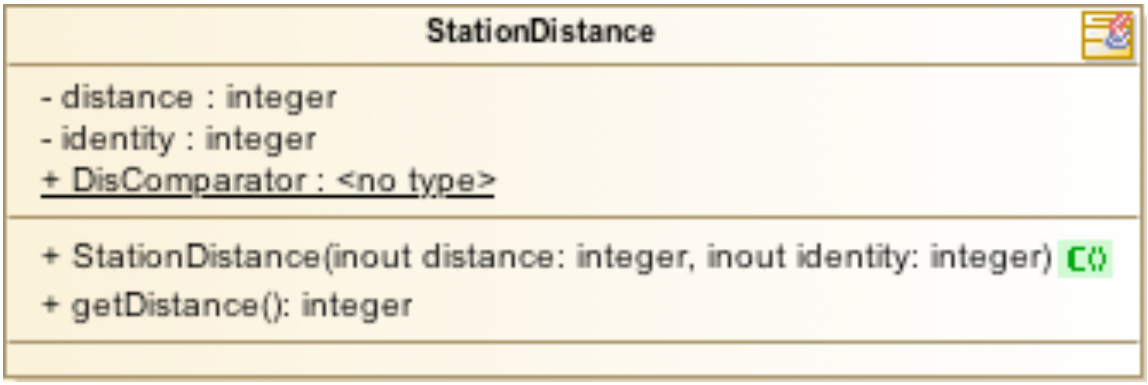


*Figure 9 StationDistance Class diagram*

Diagramme détaillé de la classe StationDistance

| Nom | Description |
|---|---|
| **StationDistance (**Inout **distance integer,**Inout **identity integer)** | Constructeur prenant la distance et l'identité de la station repérée |
| **integer getDistance ()** | Opération de nature «get» |
| **integer getIdentity ()** | Opération de nature «get» |

*Table 18 Opérations de la classe "StationDistance"*

| Nom | Description |
|---|---|
| **distance : [1..1] integer** | Attribut de la distance entre deux stations |
| **identity : [1..1] integer** | Attribut de l'identité de la station repérée |
| **DisComparator : [1..1]** | Comparator comparant la distance |

*Table 19 Attributs de la classe "StationDistance"*

## 6.4 Classe "State"

*provient de Package ProjetVelibDJMZ2.Data*

Stéréotypes: JavaClass

La classe State garde les informations d'un état d'une station correspondante à un instant. Les information sont surtout la quantité de vélos et places libres dans une station et la date correspondante. Quand l'utilisateur fournisse des données, le logiciel va créer un seul State - l'état initial - pour chaque station. Ensuite l'exécution d'une simulation va ajouter une série de nouveaux états pour chaque station. Chaque Trip exécuté va créer 2 nouveau états, l'un à la station initial, l'autre à la station terminal sauf pour des Trips annulés. Les information dans cette classe permettent l'analyse du comportement du système de Vélib.

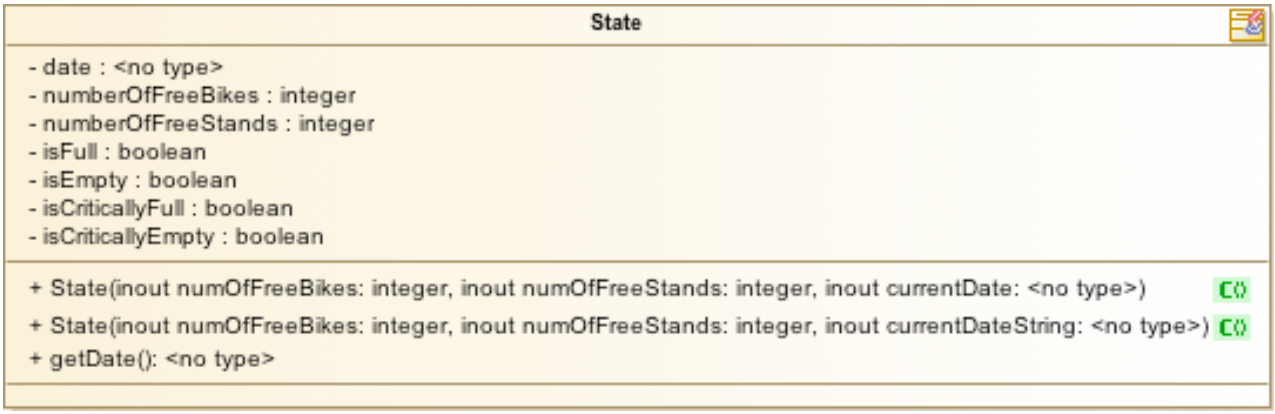


*Figure 10 State Class diagram*

Diagramme détaillé de la classe State

| Nom | Description |
|---|---|
| **State (**Inout **numOfFreeBikes integer,**Inout **numOfFreeStands integer,**Inout **currentDate )** | Constructeur prenant nombre de vélos libres, nombre d'embarquements libres, et Date |
| **State (**Inout **numOfFreeBikes integer,**Inout **numOfFreeStands integer,**Inout **currentDateString )** | Constructeur prenant nombre de vélos libres, nombre d'embarquements libres, et Date en String |
| **getDate ()** | Opération de nature «get» |
| **integer getNBikes ()** | Opération de nature «get» |
| **integer getNStands ()** | Opération de nature «get» |
| **boolean isFull ()** | Opération de nature «get» |
| **boolean isEmpty ()** | Opération de nature «get» |
| **boolean isCriticallyFull ()** | Opération de nature «get» |
| **boolean isCriticallyEmpty ()** | Opération de nature «get» |
| **setDate (**In **p1 )** | Opération de nature «set» pour mettre une nouvelle date |
| **toString ()** | Opération servant à l'affichage |
| **main (**Inout **args )** | Opération de test |

*Table 20 Opérations de la classe "State"*

| Nom | Description |
|---|---|
| **date : [1..1]** | Attribut de la date de l'état |
| **numberOfFreeBikes : [1..1] integer** | Attribut de nombre des vélos libres de l'état |
| **numberOfFreeStands : [1..1] integer** | Attribut de nombre des embarquements libres de l'état |
| **isFull : [1..1] boolean** | Attribut boolean signifiant si la station est pleine |
| **isEmpty : [1..1] boolean** | Attribut boolean signifiant si la station est vide |
| **isCriticallyFull : [1..1] boolean** | Attribut boolean signifiant si la staion est quatilifé d'être pris dans le nouveau comportement |
| **isCriticallyEmpty : [1..1] boolean** | Attribut boolean signifiant si la staion est quatilifé d'être pris dans le nouveau comportement |

*Table 21 Attributs de la classe "State"*

## 6.5 Classe "StationExtendedDynamic"

*provient de Package ProjetVelibDJMZ2.Data*

Hérite de: StationStatic

Stéréotypes: JavaClass

Pareil fonction avec StationStatic, celle-ci porte des informations dynamiques, les états et l'ouverture de stations. Vu qu'on ne change pas l'état d'ouverture après la lecture des fichiers, on a mis « isOpen » directement comme l'attribut de classe.

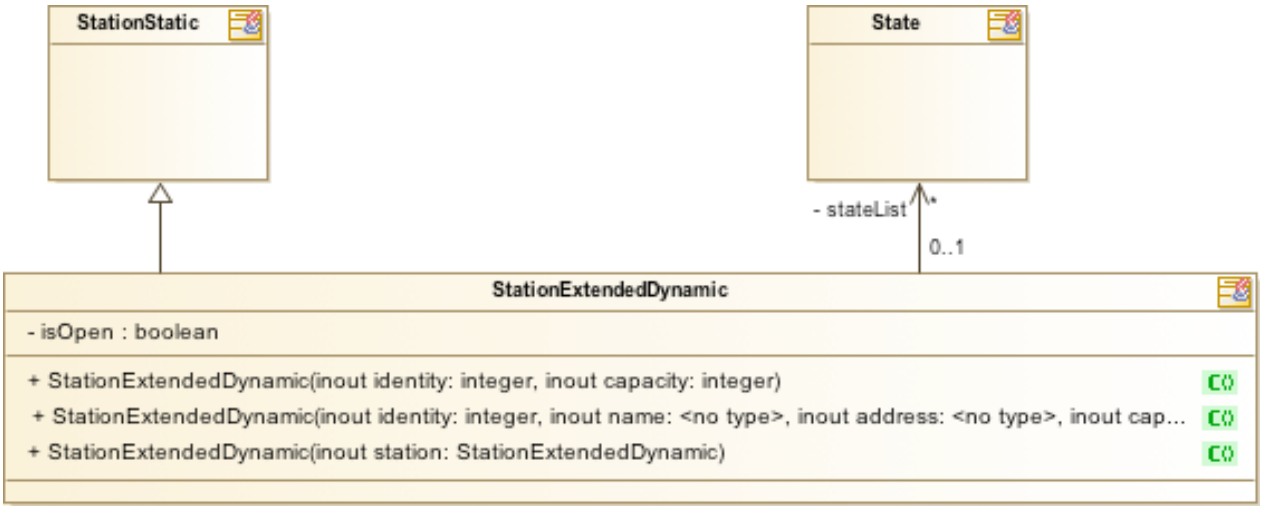


*Figure 11 StationExtendedDynamic Class diagram*

Diagramme détaillé de la classe StationExtendedDynamic

| Nom | Description |
|---|---|
| **StationExtendedDynamic (**Inout **identity integer,**Inout **capacity integer)** | Constructeur des informations prilimitaires |
| **StationExtendedDynamic (**Inout **identity integer,**Inout **name ,**Inout **address ,**Inout **capacity integer,**Inout **longitude** | Constructeur des informations complètes |

| Nom | Description |
|---|---|
| **double,**Inout **latitude double)** | |
| **StationExtendedDynamic (**Inout **station StationExtendedDynamic)** | Constructeur de copying |
| **takeBike (**Inout **date )** | Opération prenant un vélo en utilisant la date |
| **takeBike (**Inout **trip Trip)** | Opération prenant un vélo en utilisant le Trip |
| **returnBike (**Inout **date )** | Opération retournant un vélo en utilisant la date |
| **returnBike (**Inout **trip Trip)** | Opération retournant un vélo en utilisant le Trip |
| **deleteLatestState ()** | Opération supprimant la dernière état |
| **setPrimaryState (**Inout **primaryState State)** | Opération de nature «set» prenant une l'état premier pour initialiser l'état de la station |
| **setIsOpen (**Inout **isOpen boolean)** | Opération de nature «set» prenant une l'état d'ouverture pour initialiser l'état de la station |
| **State getStateList ()** | Opération de nature «get» |
| **integer getNumberOfStates ()** | Opération de nature «get» |
| **State getLatestState ()** | Opération de nature «get» |
| **State getState (**Inout **n integer)** | Opération de nature «get» |
| **boolean isOpen ()** | Opération de nature «get» |
| **toString ()** | Opération servant à l'affichage |
| **main (**Inout **args )** | Opération de test |
| **clearStates ()** | Opération supprimant tous les état |

*Table 22 Opérations de la classe "StationExtendedDynamic"*

| Nom | Description |
|---|---|
| **isOpen : [1..1] boolean** | Attribut de l'ouverture de la station |

*Table 23 Attributs de la classe "StationExtendedDynamic"*

| Nom | Description |
|---|---|
| **->stateList : [0..*]** State | ArrayList des états de la station |

*Table 24 Associations de la classe "StationExtendedDynamic"*

## 6.6 Classe "StationStatic"

*provient de Package ProjetVelibDJMZ2.Data*

Stéréotypes: JavaClass

Vu qu'il y a trois types des informations dans la classe Station, on a mis en œuvre d'une sorte d'hiérarchie de la « évolution » de Station. La StationStatic est celui de première démarche qui possède que les données statiques.

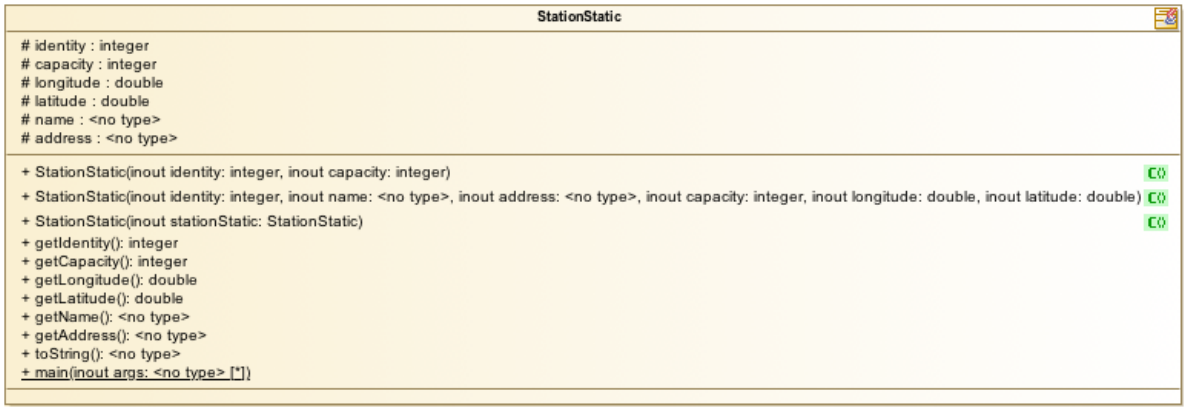


*Figure 12 StationStatic Class diagram*

| Nom | Description |
|---|---|
| **StationStatic (Inout identity integer,Inout capacity integer)** | Constructeur des informations prilimitaires |
| **StationStatic (Inout identity integer,Inout name ,Inout address ,Inout capacity integer,Inout longitude double,Inout latitude double)** | Constructeur des informations complètes |
| **StationStatic (Inout stationStatic StationStatic)** | Constructeur de copying |
| **integer getIdentity ()** | Opération de nature «get» |
| **integer getCapacity ()** | Opération de nature «get» |
| **double getLongitude ()** | Opération de nature «get» |
| **double getLatitude ()** | Opération de nature «get» |
| **getName ()** | Opération de nature «get» |
| **getAddress ()** | Opération de nature «get» |
| **toString ()** | Opération servant à l'affichage |
| **main (Inout args )** | |

*Table 25 Opérations de la classe "StationStatic"*

| Nom | Description |
|---|---|
| **identity : [1..1] integer** | |
| **capacity : [1..1] integer** | |
| **longitude : [1..1] double** | |
| **latitude : [1..1] double** | |
| **name : [1..1]** | |
| **address : [1..1]** | |

*Table 26 Attributs de la classe "StationStatic"*

# 7 Package "Simulation"

*provient de Package* *ProjetVelibDJMZ2*

Stéréotypes : JavaPackage

Ce paquetage contient les classes qui sont en charge de la manipulation des Données pour modéliser le comportement du système Vélib réel

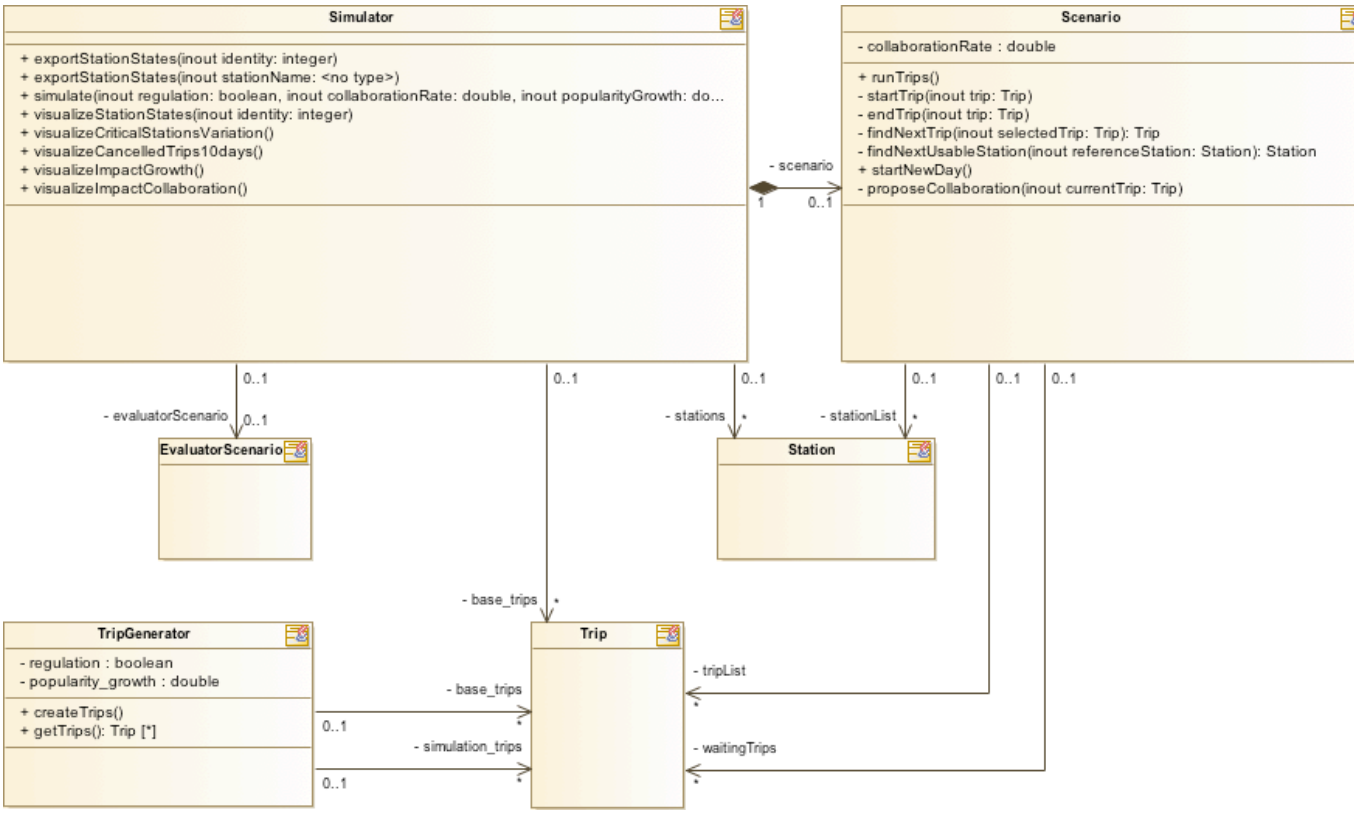


*Figure 13 Simulation Class diagram*

| Nom | Résumé |
|---|---|
| Scenario | Cette classe manipule les données réelles d'une manière spécifié par l'utilisateur pour obtenir des données traitées |
| Simulator | Classe coordonnant classe Scenario et le package Evaluation |
| TripGenerator | La classe TripGenerator est capable de créer des nouvelles Trips pour faire un pré-traitement des données |

*Table 27 Classes appartenant au package "Simulation"*

## 7.1 Classe "Scenario"

*provient de Package ProjetVelibDJMZ2.Simulation*

Stéréotypes: JavaClass

La classe principale est la classe *Scenario* qui exécute des *Trips*, c'est-à-dire que cette classe manipule les données d'une manière spécifié par l'utilisateur pour obtenir des données traitées. Une instance de cette classe avec les instances des classes du package *Data* liées correspondent à une simulation. La fonction principale est la méthode *runTrips()*. Pour faire une simulation il faut fournir une instance de *Scenario* avec un ArrayList de *Stations* (avec au moins un état (*State*) initial) et un ArrayList de Trips et appeler *runTrips()*. La série de méthodes de type private dans cette classe réalisent des sous-fonctions de *runTrips()*.

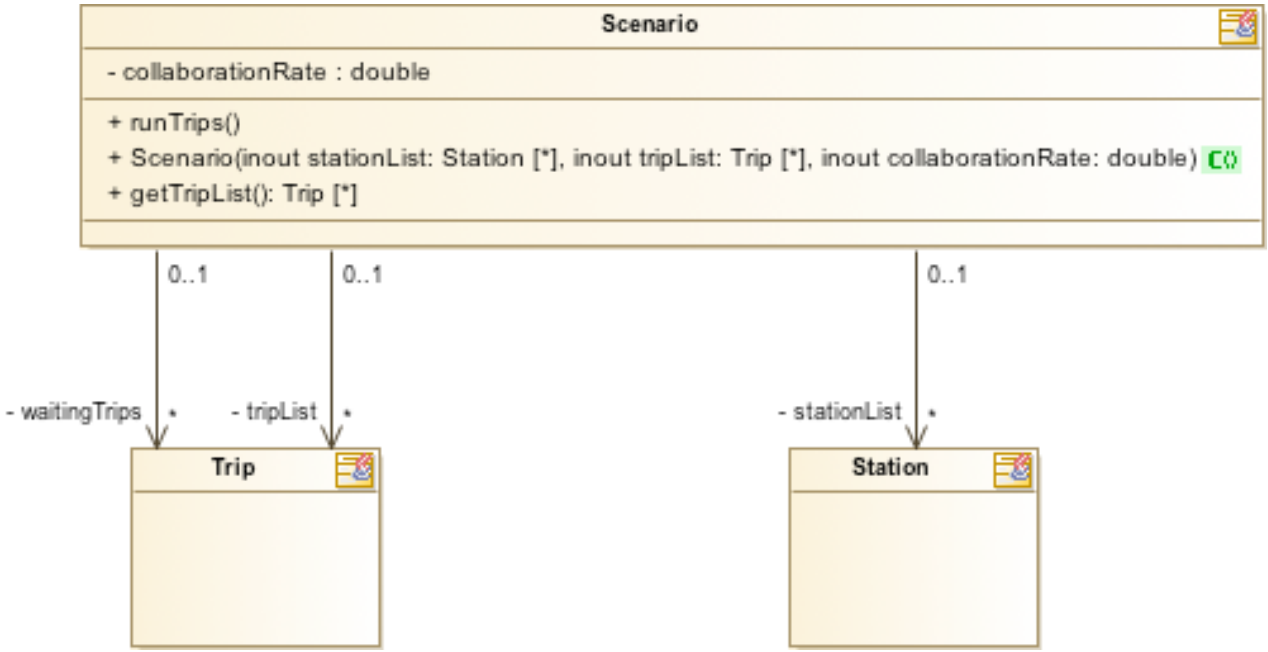


*Figure 14 Scenario Class diagram*

| Nom | Description |
|---|---|
| **runTrips ()** | Opération principale manipulant des Données réelles |
| **Scenario (**Inout **stationList Station,**Inout **tripList Trip,**Inout **collaborationRate double)** | Constructeur |
| **Trip getTripList ()** | retourne l'attribut TripList |
| **Station getStationList ()** | retourne l'attribut StationList |
| **startTrip (**Inout **trip Trip)** | Opération exécutant le début d'un Trip |
| **endTrip (**Inout **trip Trip)** | Opération exécutant le fin d'un Trip |
| **Trip findNextTrip (**Inout **selectedTrip Trip)** | Opération comparant les Dates des Trips pour trouver le prochain Trip à exécuter |
| **Station findNextUsableStation (**Inout **referenceStation Station)** | Opération pour initialisez findNextUsableStation (Station, int) avec une valeur int=1 par defaut |
| **Station findNextUsableStation (**Inout **referenceStation Station,**Inout **iteration integer)** | Opération trouvant une Station proche et vide |
| **main (**Inout **args )** | Méthode main pour tester la classe Scenario |

| Nom | Description |
|---|---|
| **proposeCollaboration (**Inout **currentTrip Trip)** | Opération trouvant une Station proche et vide basée sur le collaborationRate |
| **startNewDay ()** | Opération reinitialisant le dernier State d'une simulation comme premier State en supprimant tous les autres States |

*Table 28 Opérations de la classe "Scenario"*

| Nom | Description |
|---|---|
| **collaborationRate : [1..1] double** | valeur de type double qui peut prendre des valeurs entre (0 ;1) correspondant à une collaboration des utilisateurs de 0 ou 100% |

*Table 29 Attributs de la classe "Scenario"*

| Nom | Description |
|---|---|
| **->tripList : [0..*]** Trip | ArrayList qui garde les données des Trips (dans l'ordre de startDate) |
| **->stationList : [0..*]** Station | ArrayList qui garde les données des Stations |
| **->waitingTrips : [0..*]** Trip | ArrayList dynamique stockant les Trips qui ont commencé mais pas encore fini |

*Table 30 Associations de la classe "Scenario"*

## 7.2 Classe "Simulator"

*provient de Package ProjetVelibDJMZ2.Simulation*

Stéréotypes: JavaClass

Classe coordonnant classe Scenario et le package Evaluation

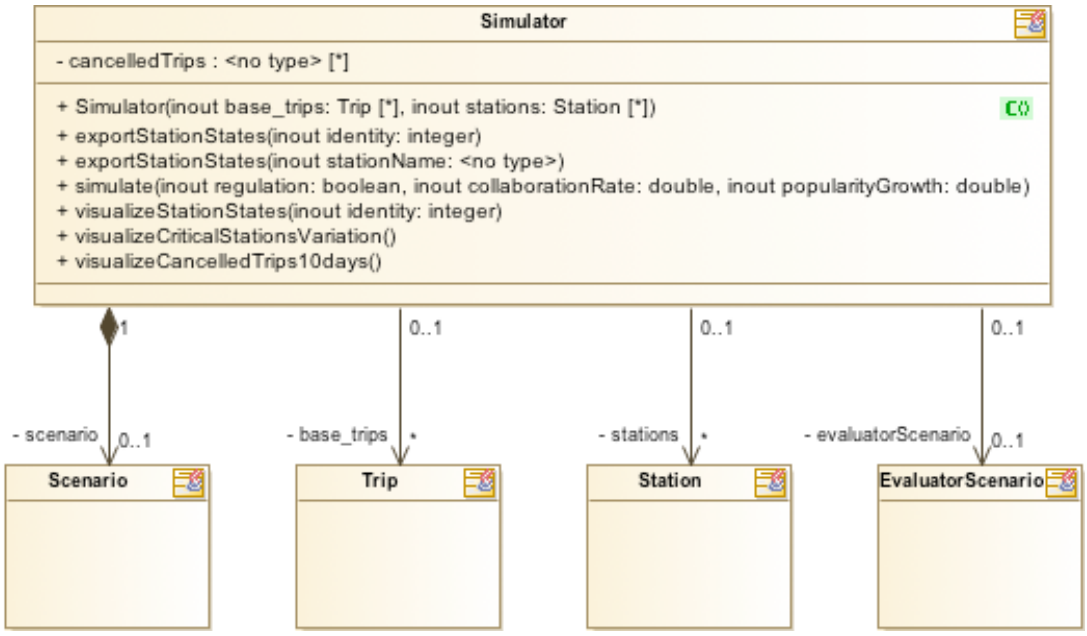


*Figure 15 Simulator Class diagram*

| Nom | Description |
|---|---|
| **Simulator (**Inout **base_trips Trip,**Inout **stations Station)** | initialize l'objet avec les paramètres de simulation et les listes de trajets et stations |
| **exportStationStates (**Inout **identity integer)** | sauvegarde tous les états d'une station. Le fichier sera sauvegardé dans "output/states_" + stationId + ".csv" |
| **exportStationStates (**Inout **stationName )** | |
| **simulate (**Inout **regulation boolean,**Inout **collaborationRate double,**Inout **popularityGrowth double)** | execute la simulation |
| **visualizeStationStates (**Inout **identity integer)** | montre un graph avec la variation de vélos disponibles dans une station. Identity est la identité de la station choisie |
| **visualizeCriticalStationsVariation ()** | montre dans un graph la variation de station vides ou pleines pendant la journée |
| **visualizeCancelledTrips10days ()** | montre la pourcentage de trajets annulées dans un période de 10 jours |
| **visualizeImpactGrowth ()** | montre un graph qui résume le comportement du système avec la variation de la croissance de popularité |
| **visualizeImpactCollaboration ()** | montre un graph qui résume le comportement du système avec la variation de la taux de collaboration des usageurs |
| **simulate10days (**Inout **regulation boolean,**Inout **collaborationRate double,**Inout **popularityGrowth double)** | fait la simulation d'une periode 10 days |

*Table 31 Opérations de la classe "Simulator"*

| Nom | Description |
|---|---|
| **cancelledTrips : [0..*]** | |

*Table 32 Attributs de la classe "Simulator"*

| Nom | Description |
|---|---|
| **->scenario : [0..1] Scenario** | |
| **->base_trips : [0..*] Trip** | |
| **->stations : [0..*] Station** | |
| **->evaluatorScenario : [0..1] EvaluatorScenario** | |

*Table 33 Associations de la classe "Simulator"*

## 7.3 Classe "TripGenerator"

*provient de Package ProjetVelibDJMZ2.Simulation*

Stéréotypes: JavaClass

La classe *TripGenerator* est capable de créer des nouvelles *Trips* pour faire un pré-traitement des données. On a besoin de ses fonctions pour simuler une croissance de popularité. Cette classe utilise les T*rips* des données réelles et fait une copie de certaines *Trips* aléatoirement.

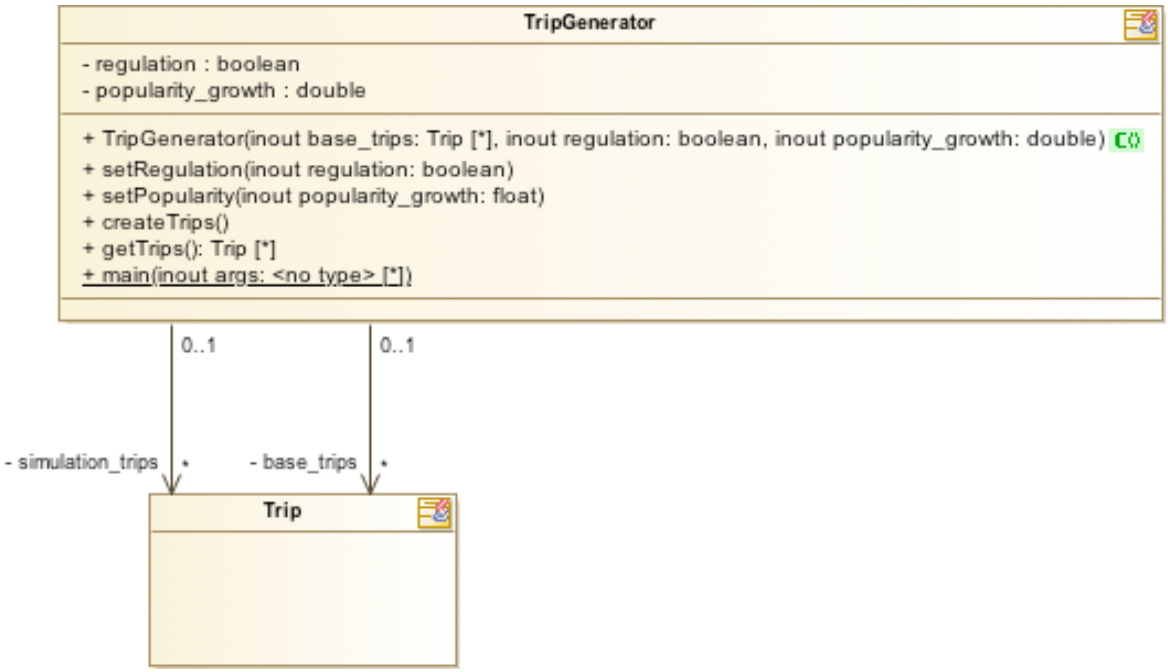


*Figure 16 TripGenerator Class diagram*

| Nom | Description |
|---|---|
| **TripGenerator (**Inout **base_trips Trip,**Inout **regulation boolean,**Inout **popularity_growth double)** | Constructeur |
| **setRegulation (**Inout **regulation boolean)** | Opération changeant le valeur de regulation |
| **setPopularity (**Inout **popularity_growth float)** | Opération changeant le valeur de popularity_growth |
| **createTrips ()** | Opération créant des nouvelles Trips basé sur le popularity_growth |
| **Trip getTrips ()** | retourne l'attribut simulationTrips |
| **main (**Inout **args )** | Méthode main pour tester la classe TripGenerator |

*Table 34 Opérations de la classe "TripGenerator"*

| Nom | Description |
|---|---|
| **regulation : [1..1] boolean** | Boolean représentant l'existence de régulation |
| **popularity_growth : [1..1] double** | double qui prend des valeurs entre (0 ;∞) représentant une croissance de popularité (0 ≙ 0%, 1 ≙ 100%) |

*Table 35 Attributs de la classe "TripGenerator"*

| Nom | Description |
|---|---|
| **->base_trips : [0..*]** Trip | Arraylist des données brut de Trips |
| **->simulation_trips : [0..*]** Trip | ArrayList des Trips incluant des Trips généré par cette classe |

*Table 36 Associations de la classe "TripGenerator"*

# 8 Package "Evaluation"

*provient de Package* *ProjetVelibDJMZ2*

Stéréotypes : JavaPackage

Package pour faire une analyze de la variations des états des stations. Le but est de fournir des visualizations utiles et la possibilité d'exporter des données d'une simulation.

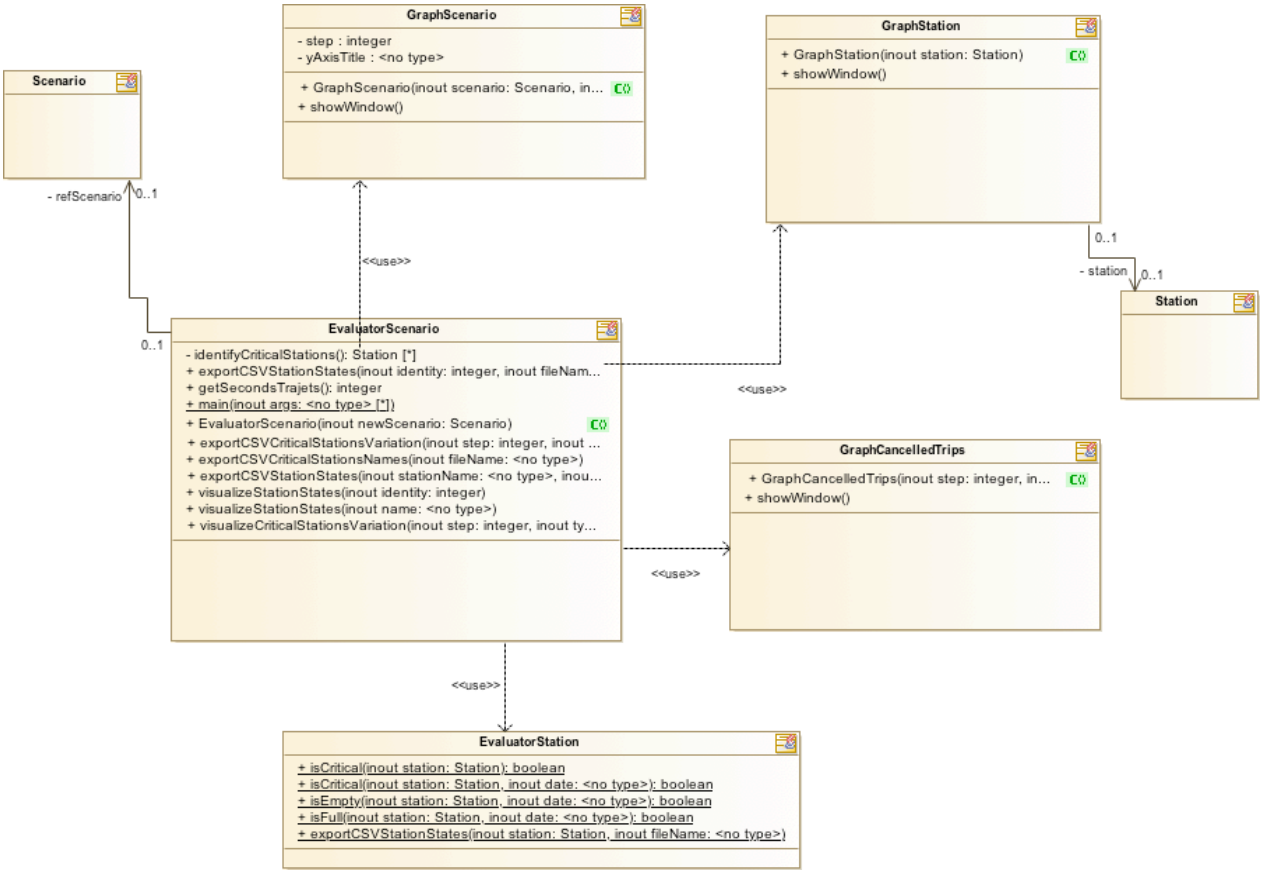


***Figure 17 Evaluation Class diagram***

| Nom | Résumé |
|---|---|
| EvaluatorScenario | Classe pour faire des analyzes d'un Scenario après la validation de ses trips. Cette classe peut exporter et montrer des données. |
| EvaluatorStation | classe statique pour garder un ensemble de fonctions qui seront utilisé dans la classe |
| GraphScenario | Classe pour montrer la variation de stations vides et pleines pendant la journée |
| GraphStation | Classe pour montrer la variation de vélos dans une station pendant la journée. |
| GraphCancelledTrips | Classe pour montrer la variation de taux d'annulation des trajets par rapport à la variation d'un paramètre dans la simulation |

***Table 37 Classes appartenant au package "Evaluation"***

| Nom | Résumé |
|---|---|
| Data | : Paquetage qui contient toutes les formes de données (Station, États des stations, Trip...) en question |

*Table 38 Eléments importés par le package "Evaluation"*

## 8.1 Classe "EvaluatorScenario"

*provient de Package ProjetVelibDJMZ2.Evaluation*

Stéréotypes: JavaClass

Classe pour faire des analyzes d'un Scenario après la validation de ses trips. Cette classe peut exporter et montrer des données

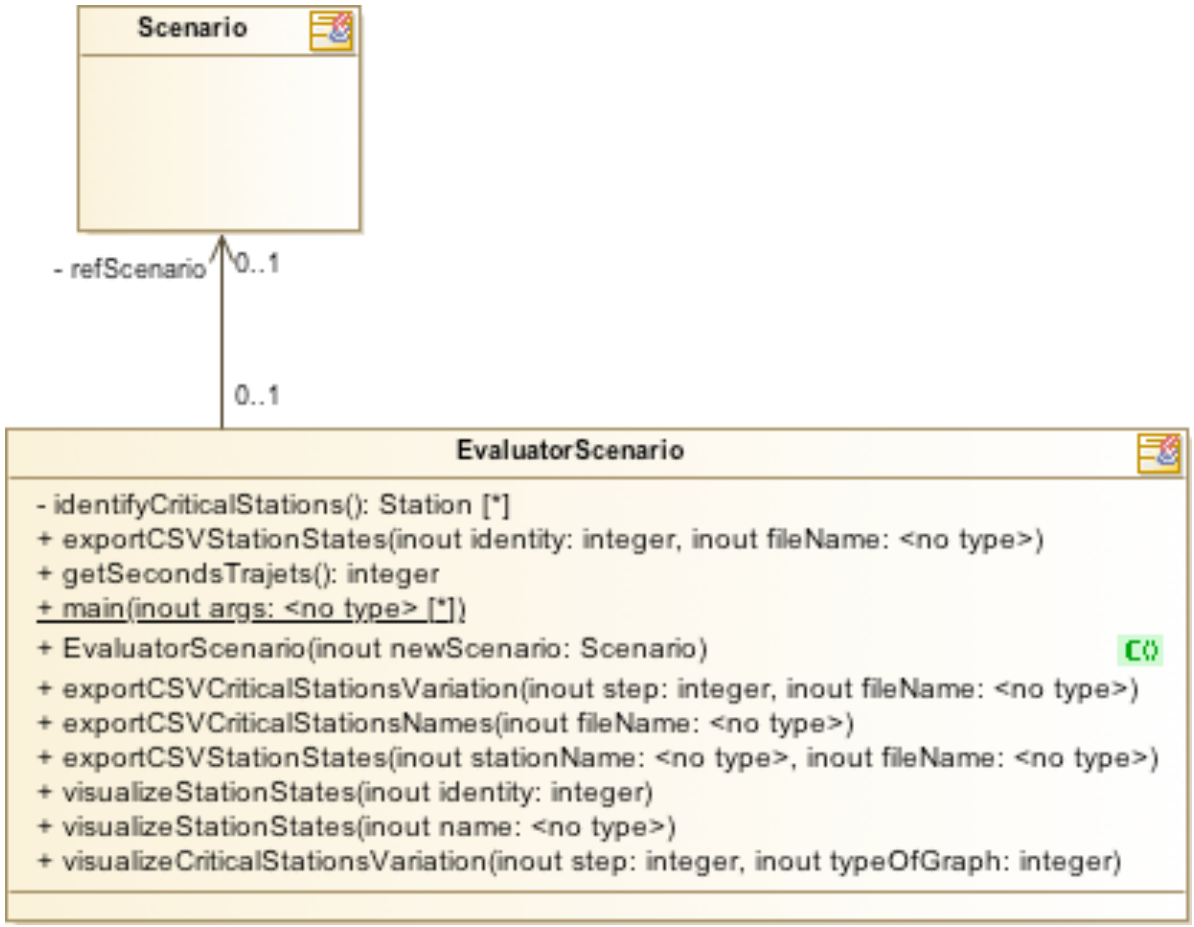


*Figure 18 EvaluatorScenario Class diagram*

Diagramme détaillé de la classe EvaluatorScenario

| Nom | Description |
|---|---|
| **Station identifyCriticalStations ()** | identifier tous les stations critiques, tous les stations qui sont restées vides ou pleines pendant au moins 4 heures dans une journée |

| Nom | Description |
|---|---|
| **exportCSVStationStates (**Inout **identity integer,**Inout **fileName )** | écrit dans un fichier CSV la variation du nombre de stations vides ou pleines pendant la journée. Le nom du fichier sera fileName. Le step est la variation de temps entre deux points dans le graph. |
| **integer getSecondsTrajets ()** | retourne la somme de la duration de tous les trajets validés |
| **main (**Inout **args )** | |
| **EvaluatorScenario (**Inout **newScenario Scenario)** | initialize l'objet avec le scenario à analyzer |
| **exportCSVCriticalStationsVariation (**Inout **step integer,**Inout **fileName )** | écrit dans un fichier CSV la variation du nombre de stations vides ou pleines pendant la journée. Le nom du fichier sera fileName. Le step est la variation de temps entre deux points dans le graph. |
| **exportCSVCriticalStationsNames (**Inout **fileName )** | écrit dans un fichier CSV tous les noms des stations critiques.Le nom du fichier sera fileName + .csv |
| **exportCSVStationStates (**Inout **stationName ,**Inout **fileName )** | écrit dans un fichier CSV tous les états de la station d'identité égal à id. Le nom du fichier sera fileName + .csv |
| **visualizeStationStates (**Inout **identity integer)** | Montre dans un graph la variation de vélos dans une station d'identité égal à id. |
| **visualizeStationStates (**Inout **name )** | Montre dans un graph la variation de vélos dans une station. |
| **visualizeCriticalStationsVariation (**Inout **step integer,**Inout **typeOfGraph integer)** | montre dans un graph la variation du nombre de stations vides ou pleines pendant la journée. |
| **double getCancelledTrips ()** | donne le pourcentage de trajets annulés. |

*Table 39 Opérations de la classe "EvaluatorScenario"*

| Nom | Description |
|---|---|
| **->refScenario : [0..1]** Scenario | scenario à analyzer |

*Table 40 Associations de la classe "EvaluatorScenario"*

## 8.2 Classe "EvaluatorStation"

*provient de Package ProjetVelibDJMZ2.*Evaluation

Stéréotypes: JavaClass

Classe statique pour garder un ensemble de fonctions qui seront utilisé dans la classe EvaluatorScenario

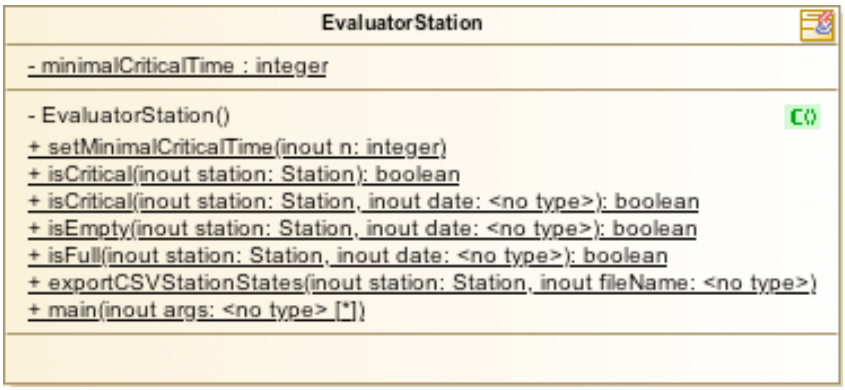


*Figure 19 EvaluatorStation Class diagram*

| Nom | Description |
|---|---|
| **EvaluatorStation ()** | |
| **setMinimalCriticalTime (**Inout **n integer)** | configure le critère de stations critique pour la classe |
| **boolean isCritical (**Inout **station Station)** | vérifie si la station est critique selon les critères définis. |
| **boolean isCritical (**Inout **station Station,**Inout **date )** | verifie si la station est vide ou pleine dans une date. |
| **boolean isEmpty (**Inout **station Station,**Inout **date )** | vérifie si la station est vide dans une date. |
| **boolean isFull (**Inout **station Station,**Inout **date )** | vérifie si la station est pleine dans une date |
| **exportCSVStationStates (**Inout **station Station,**Inout **fileName )** | écrit dans un fichier CSV tous les états de la station d'identité égal à id. Le nom du fichier sera fileName + stationId + .csv |
| **main (**Inout **args )** | |

*Table 41 Opérations de la classe "EvaluatorStation"*

| Nom | Description |
|---|---|
| **minimalCriticalTime : [1..1] integer** | temps minimal(en miliseconds) que une stations doit rester vide ou pleine pour être consideré critique. |

*Table 42 Attributs de la classe "EvaluatorStation"*

## 8.3 Classe "GraphScenario"

*provient de Package ProjetVelibDJMZ2.Evaluation*

Stéréotypes: JavaClass

Classe pour montrer la variation de stations vides et pleines pendant la journée


*Figure 20 GraphScenario Class diagram*

| Nom | Description |
|---|---|
| **GraphScenario (**Inout **scenario Scenario,**Inout **step integer,**Inout **type integer)** | initialize la classe avec le scénario qui sera utilisé, l'espace entre deux points(step) et le type du graph(1 = stations vides, 2 = stations pleines, 3 = l'ensemble des deux) |
| **createDataset ()** | configure les données pour le graph. Utilise les stations vides ou pleines. |
| **createDatasetEmpty ()** | configure les données pour le graph. |
| **createDatasetFull ()** | configure les données pour le graph. Utilise les stations pleines |
| **showWindow ()** | crée la fenêtre et la montre |
| **createChart (**In **dataset )** | configure les données pour le graph. |
| **main (**In **args )** | |

*Table 43 Opérations de la classe "GraphScenario"*

| Nom | Description |
|---|---|
| **step : [1..1] integer** | space entre deux points dans l'axis X |
| **yAxisTitle : [1..1]** | le titre du axis Y |

*Table 44 Attributs de la classe "GraphScenario"*

| Nom | Description |
|---|---|
| **->scenario : [0..1]** Scenario | scenario qui sera utilisé pour le graph |

*Table 45 Associations de la classe "GraphScenario"*

## 8.4 Classe "GraphStation"

*provient de Package ProjetVelibDJMZ2.Evaluation*

Stéréotypes: JavaClass

Classe pour montrer la variation de vélos dans une station pendant la journée.

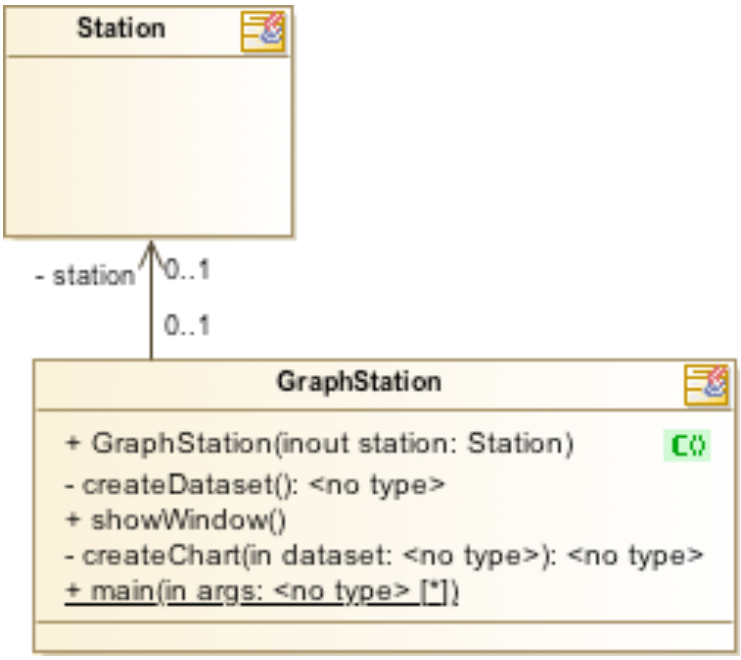


*Figure 21 GraphStation Class diagram*

| Nom | Description |
|---|---|
| **GraphStation (**Inout **station Station)** | initialize l'objet avec la station qui sera utilisée |
| **createDataset ()** | configure les données pour le graph. |
| **showWindow ()** | crée la fênetre et la montre |
| **createChart (**In **dataset )** | configure les données pour le graph. |
| **main (**In **args )** | |

*Table 46 Opérations de la classe "GraphStation"*

| Nom | Description |
|---|---|
| **->station : [0..1]** Station | station qui sera utilisée pour le graph |

*Table 47 Associations de la classe "GraphStation"*

## 8.5 Classe "GraphCancelledTrips"

*provient de Package ProjetVelibDJMZ2.Evaluation*

Stéréotypes: JavaClass

Classe pour montrer la variation de taux d'annulation des trajets par rapport à la variation d'un paramètre dans la simulation

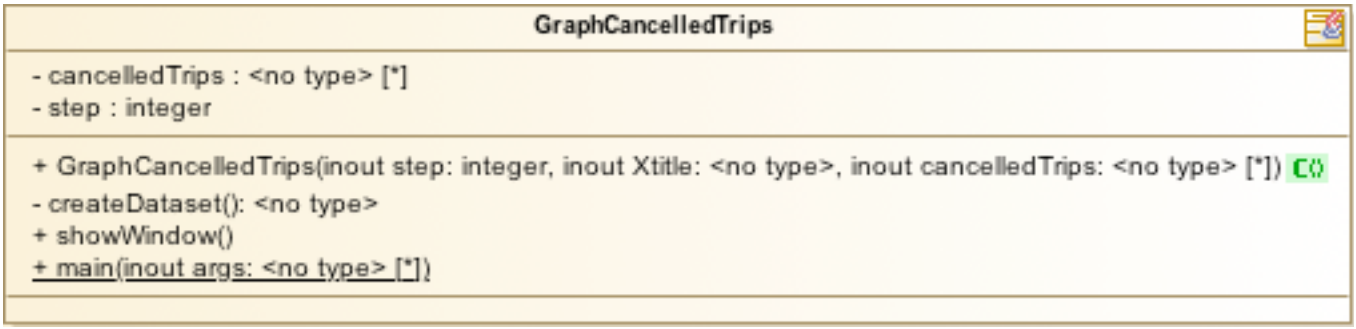


*Figure 22 GraphCancelledTrips Class diagram*

| Nom | Description |
|---|---|
| **GraphCancelledTrips (**Inout **step integer,**Inout **Xtitle ,**Inout **cancelledTrips )** | |
| **createDataset ()** | configure les données pour le graph. |
| **showWindow ()** | crée la fenêtre et la montre |
| **main (**Inout **args )** | |

*Table 48 Opérations de la classe "GraphCancelledTrips"*

| Nom | Description |
|---|---|
| **cancelledTrips : [0..*]** | données qui seront utilisées |
| **step : [1..1] integer** | space dans l'axis X entre 2 points |

*Table 49 Attributs de la classe "GraphCancelledTrips"*

# 9 Diagrammes de séquences

*Interaction "Interaction new trips"*

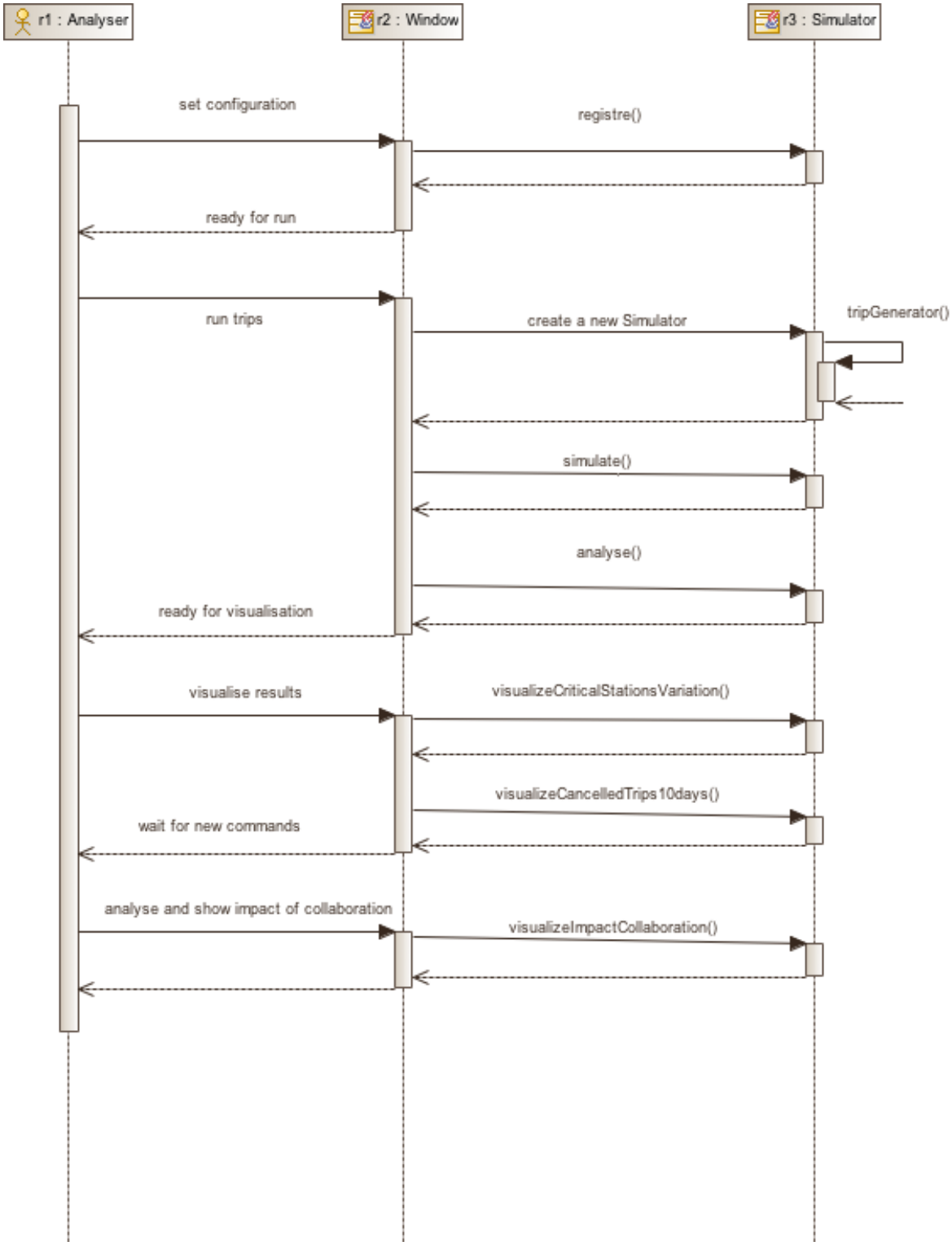


*Figure 23 Interaction Sequence diagram new trips*

## *Interaction "Interaction base scenario"*

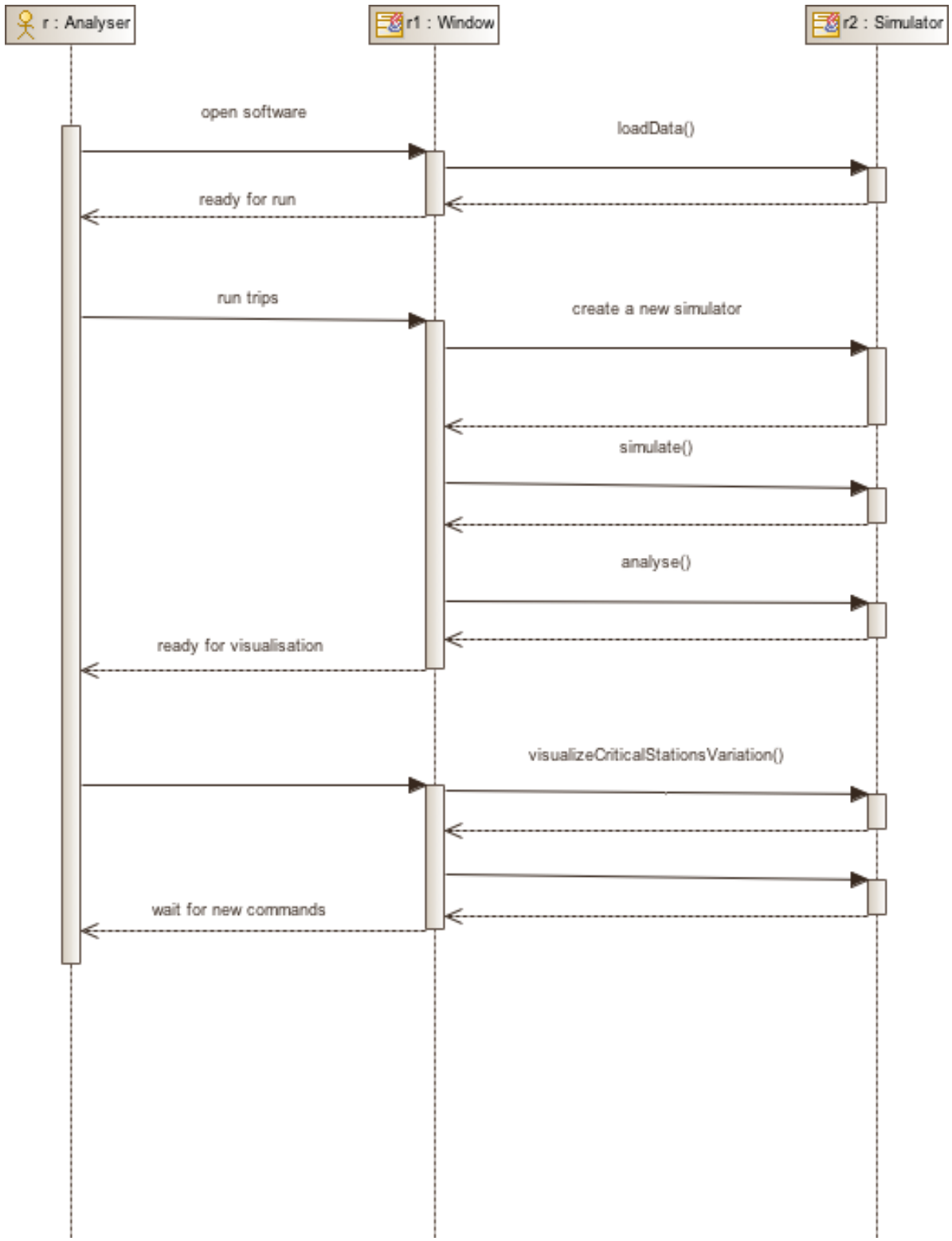


***Figure 24 Interaction Sequence diagram base scenario***

## Interaction "Interaction"

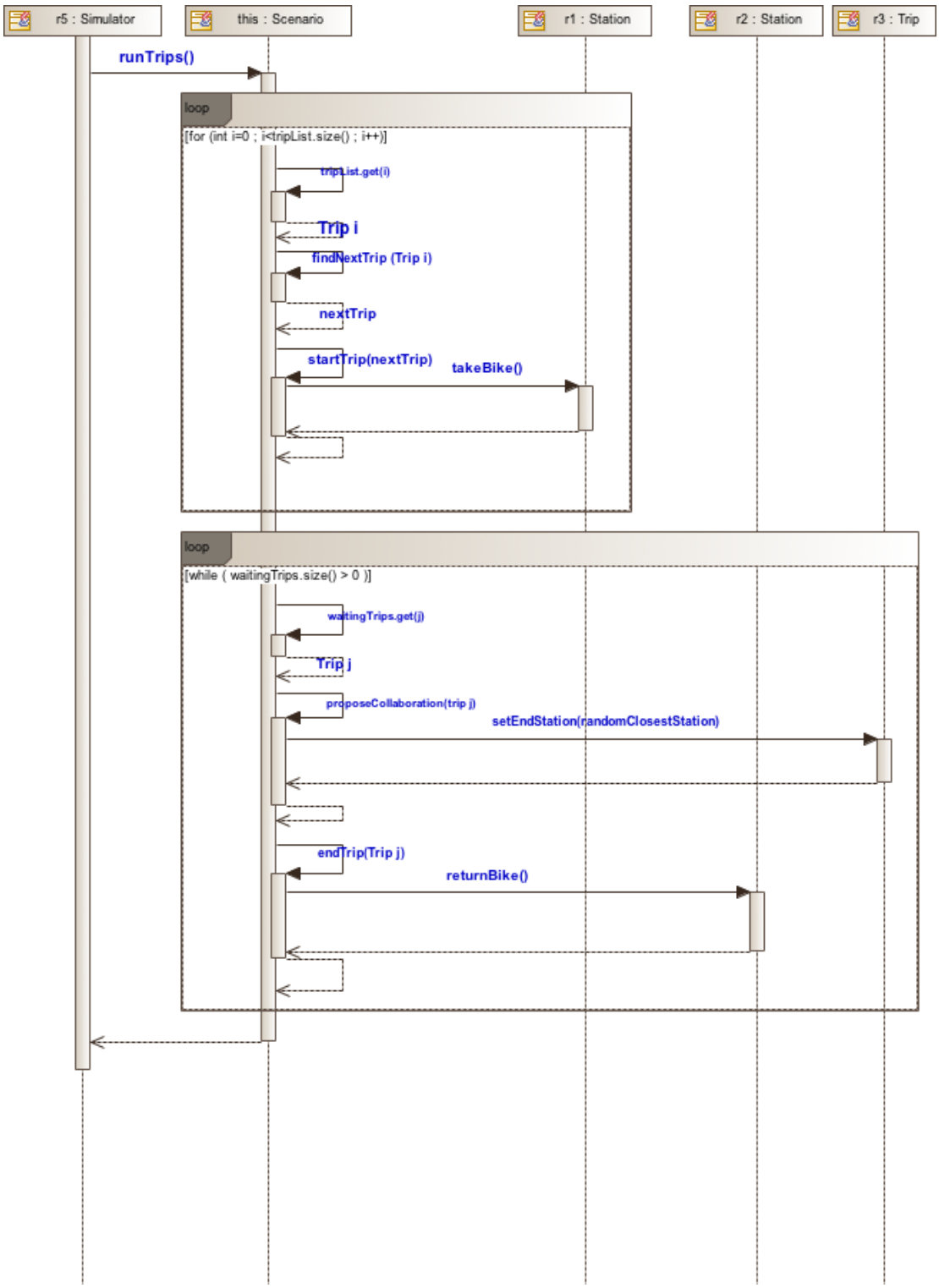


*Figure 25 Interaction Sequence diagram*

## Interaction "Interaction show impacts"

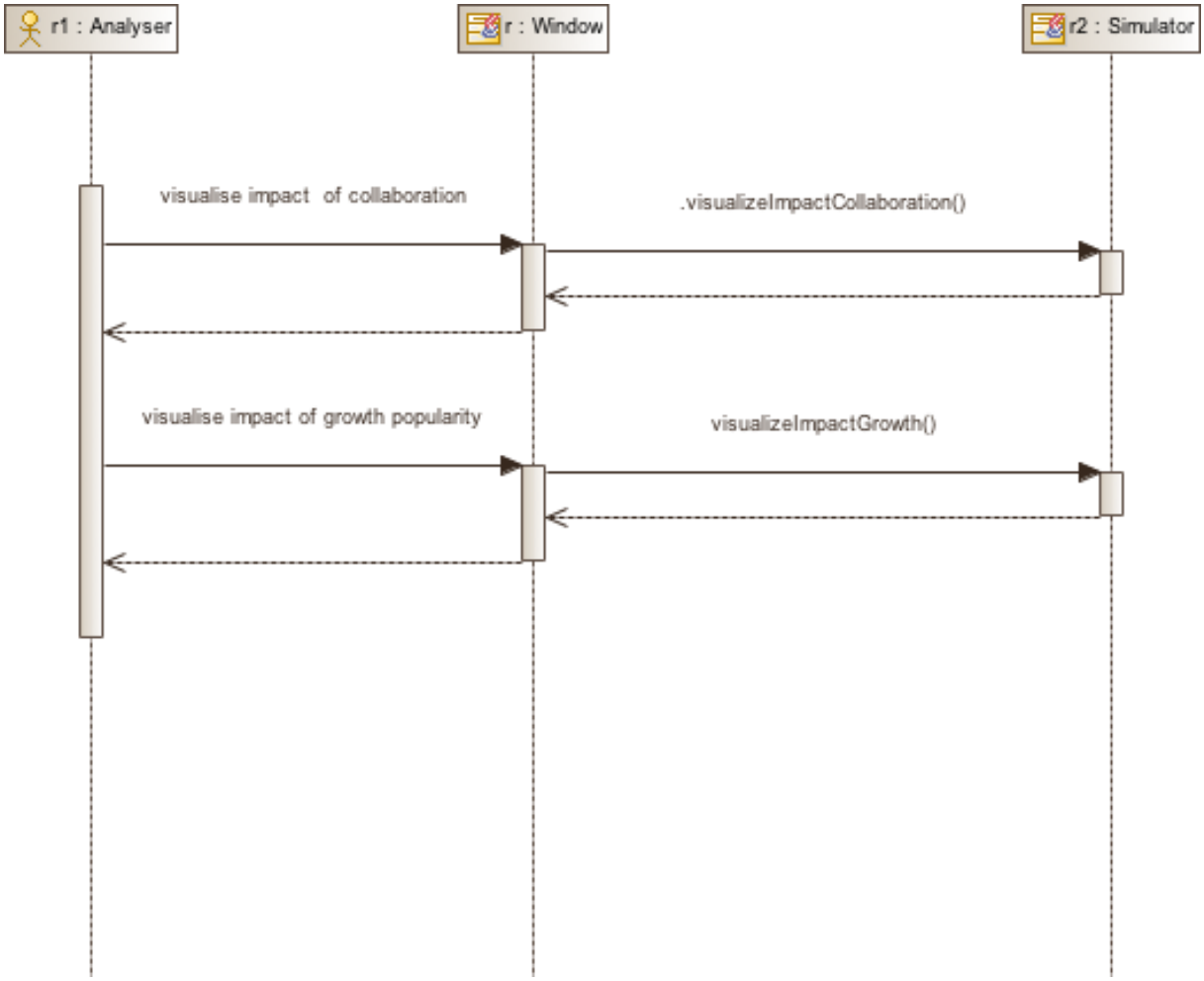


*Figure 26 Interaction Sequence diagram*