

LES ASCENCEURS DE TOLBIAC

Que doit faire notre système d'ascenseur ?



Simuler les utilisateurs



Simuler leurs demandes



Attribuer les demandes dans les différents ascenseurs



Faire rentrer utilisateurs dans l'ascenseur correspondant



Déplacer les ascenseurs



Faire sortir les utilisateurs à leurs étages

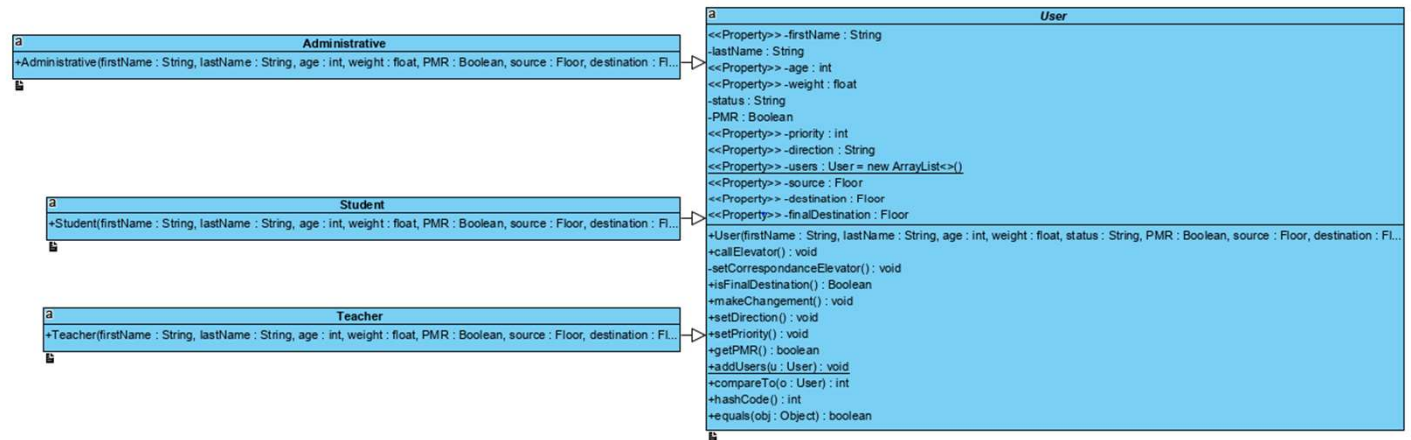


Gérer les utilisateurs devant faire un changement



Gérer la priorités des entrées et des sorties

Simuler les utilisateurs



Nous avons une classe abstraite **User**, définis par : un *nom*, un *prénom*, un *âge*, un *statut*, une *situation ou non de handicap*, un *numéro* correspondant à sa *priorité* (plus ce numéro est grand plus l'User est prioritaire) calculé par la méthode setPriority(), un étage de départ, un étage d'arrivé, un étage final (dont nous détaillerons l'utilité) ainsi qu'une direction calculé à partir de l'étage de départ et celui d'arrivé dans la méthode setDirection().

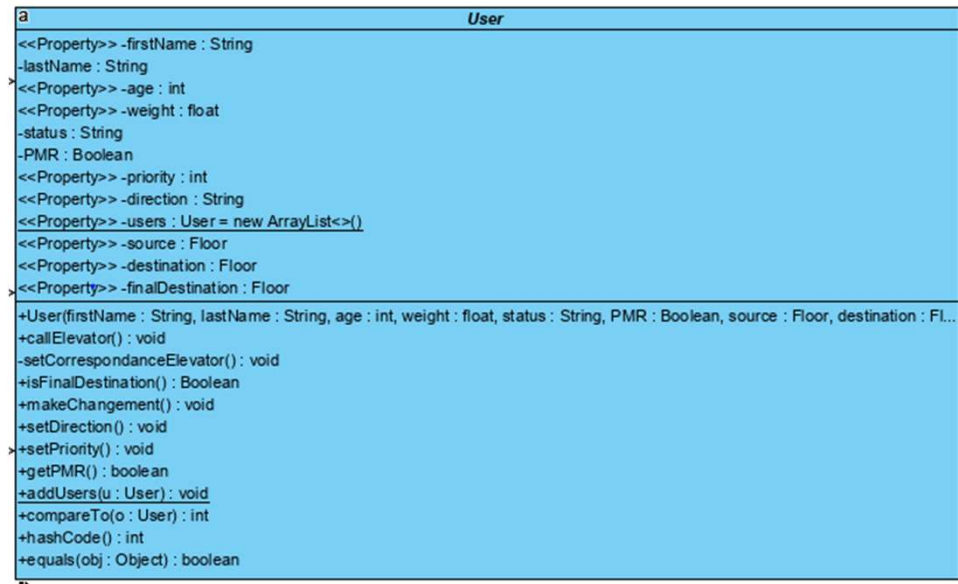
Nous avons créer les méthodes callElevator() qui crée une nouvelle demande, setCorrespondance(), isFinalDestination() et makeChangement() servent à traiter les cas d'User devant faire des correspondances,

User implémente **Comparable**. compareTo() compare les User en fonction de leur priorité.

Les méthode equals() et hashCode() on été surchargé pour utiliser des hashMap d'Users.

Les classes **Administrative**, **Student**, **Teacher** héritent de **User**. Nous redéfinissons seulement le statut de l'User. Ce statut sert à l'instanciation pour déterminer l'attribut de priorité.

Gérer les utilisateurs devant faire un changement



Dans le cas ou la couleur de départ est différente de la couleur d'arrivé :

- A l'instanciation d'un **User** nous calculons l'étage pivot à l'aide de la méthode setCorrespondance() cette étage deviens la destination et la destination final est stockée dans l'attribut *finalDestination*.
- Quand l'utilisateur arrive à sa destination pivot le test isFinalDesination() est faux et permet d'appeler la méthode makeChangement() qui mets l'User dans la queue du Floor correspondant à sa prochaine demande.

Simuler les demandes



a	Demand
<<Property>>	-direction : String
<<Property>>	-floor : Floor
+	Demand(f : Floor, dir : String)
+	hashCode() : int
+	equals(obj : Object) : boolean

UP

DOWN

La classe Demand, représente l'appel d'un utilisateur à un étage. Cette demande contient deux attributs correspondants aux deux informations envoyées par un bouton appuyé dans le système d'ascenseur :

- Un étage d'appel
- Une direction pouvant être égale à "up" ou "down"

Les méthode equals() et hashCode() on été surchargé pour utiliser des set de Demand.

Attribuer les
demandes dans
les différents
ascenseurs



a Dispatcher	
<<Property>> -listElevator : Map<String, List<Elevator>> = new HashMap<>()	
<<Property>> -demands : Demand = new LinkedHashSet<Demand>()	
+dispatch() : void	
-chooseNearestElevator(d : Demand) : Elevator	
-addDemandOnChooosen(choosen : Elevator, d : Demand) : void	
-getNbFloorToReachDemand(choosen : Elevator, d : Demand) : int	
-grabNullElevator(elevatorColor : String) : Elevator	
+addDemand(d : Demand) : void	

La classe Dispatcher contient un set de toutes les demandes remplis par la méthode `addDemand(demand)` et à pour rôle de les attribuer aux **Elevator** à travers la méthode `dispatch()`. Pour chaque demande en cours :

- Le dispatcher la donne en priorité à un **Elevator** à l'arrêt de la bonne couleur qu'il récupère à l'aide de la méthode `grabNullElevator(ElevatorColor)`.
- Si cette **Elevator** est en dessous d'une demande en monté ou au dessus d'une demande en descente on calcul le nombre d'itération (`getNbFloorToReachDemand(choosen, demand)`) sur les **Floor** à effectuer par l'**Elevator** avant de prendre des **Users** et de changer de direction. On ajoute la demande via `addDemandOnChooosen(choosen, demand)`
- Si aucun **Elevator** n'est à l'arrêt il va calculer l'**Elevator** le plus proche dont le chemin passe par la demande
- Si aucun **Elevator** ne répond à cette demande il laisse la demande pour le prochaine appel à `dispatch()`

Faire rentrer
utilisateurs dans
l'ascenseur
correspondant



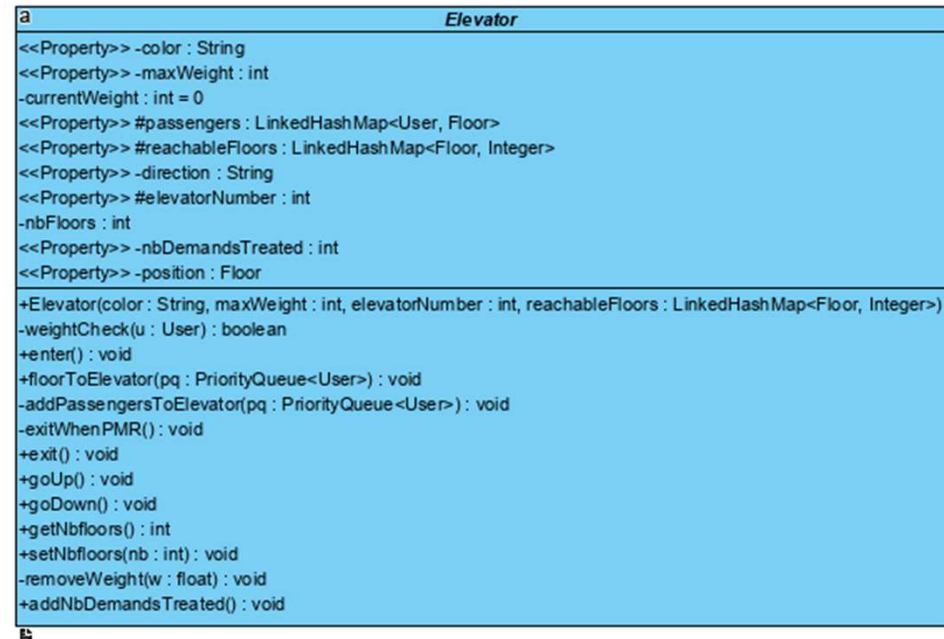
```
a
classDiagram
    class Floor {
        <<Property>> -floorNumber : int
        <<Property>> -color : String
        <<Property>> -usersUp : User = new PriorityQueue<>()
        <<Property>> -usersDown : User = new PriorityQueue<>()
        <<Property>> -previousFloor : Floor
        <<Property>> -nextFloor : Floor
        <<Property>> -floors : Floor = new HashSet<>()
        +Floor(floorNumber : int, color : String)
        +getNextFloor() : Floor
        +getPreviousFloor() : Floor
        +addUsersUp(u : User) : void
        +addUsersDown(u : User) : void
        +hashCode() : int
        +equals(obj : Object) : boolean
        +getFloor(number : int, color : String) : Floor
    }
    Floor "1" -- "1" Floor : nextFloor
    Floor "1" -- "1" Floor : previousFloor
    Floor "1" -- "1" Floor : floors
```

```
a
classDiagram
    class Elevator {
        <<Property>> -color : String
        <<Property>> -maxWeight : int
        <<Property>> -currentWeight : int = 0
        <<Property>> #passengers : LinkedHashMap<User, Floor>
        <<Property>> #reachableFloors : LinkedHashMap<Floor, Integer>
        <<Property>> -direction : String
        <<Property>> #elevatorNumber : int
        <<Property>> -nbFloors : int
        <<Property>> -nbDemandsTreated : int
        <<Property>> -position : Floor
        +Elevator(color : String, maxWeight : int, elevatorNumber : int, reachableFloors : LinkedHashMap<Floor, Integer>)
        -weightCheck(u : User) : boolean
        +enter() : void
        +floorToElevator(pq : PriorityQueue<User>) : void
        -addPassengersToElevator(pq : PriorityQueue<User>) : void
        -exitWhenPMR() : void
        +exit() : void
        +goUp() : void
        +goDown() : void
        +getNbFloors() : int
        +setNbFloors(nb : int) : void
        -removeWeight(w : float) : void
        +addNbDemandsTreated() : void
    }
    Elevator "1" -- "1" Floor : position
```

Chaque **Floor** possède deux **PriorityQueue** représentant respectivement les **Users** attendant pour monter (*userUp*) et ceux pour descendre (*userDown*). Les Users sont rangés par ordre de priorité.

La méthode enter() permet de faire rentrer les utilisateurs se trouvant à l'étage de la position de l'**Elevator** et étant dans la queue correspondant à la direction de l'**Elevator**. Elle appelle la méthode `floorToElevator(PriorityQueue)`. Tant que le *poids* (`weightChecker()`) ou la *surface* le permet.

Gérer la priorités des entrées et des sorties



les **Users** sont ajoutés par ordre de priorité (`addPassengerToElevator(PriorityQueue)`). Si il reste un PMR dans la queue qui n'a pas pu rentrer, la fonction `exitWhenPMR()` est appelée. Elle retire les utilisateurs par ordre décroissant de priorité jusqu'à que le PMR puisse rentrer.

Déplacer les ascenseurs



```
a
Floor
<<Property>> -floorNumber : int
<<Property>> -color : String
<<Property>> -usersUp : User = new PriorityQueue<>()
<<Property>> -usersDown : User = new PriorityQueue<>()
<<Property>> -previousFloor : Floor
<<Property>> -nextFloor : Floor
<<Property>> -floors : Floor = new HashSet<>()
+Floor(floorNumber : int, color : String)
+getNextFloor() : Floor
+getPreviousFloor() : Floor
+addUsersUp(u : User) : void
+addUsersDown(u : User) : void
+hashCode() : int
+equals(obj : Object) : boolean
+getFloor(number : int, color : String) : Floor
```

```
a
Elevator
<<Property>> -color : String
<<Property>> -maxWeight : int
-currentWeight : int = 0
<<Property>> #passengers : LinkedHashMap<User, Floor>
<<Property>> #reachableFloors : LinkedHashMap<Floor, Integer>
<<Property>> -direction : String
<<Property>> #elevatorNumber : int
-nbFloors : int
<<Property>> -nbDemandsTreated : int
<<Property>> -position : Floor
+Elevator(color : String, maxWeight : int, elevatorNumber : int, reachableFloors : LinkedHashMap<Floor, Integer>)
-weightCheck(u : User) : boolean
+enter() : void
+floorToElevator(pq : PriorityQueue<User>) : void
+addPassengersToElevator(pq : PriorityQueue<User>) : void
+exitWhenPMR() : void
+exit() : void
+goUp() : void
+goDown() : void
+getNbFloors() : int
+setNbFloors(nb : int) : void
+removeWeight(w : float) : void
+addNbDemandsTreated() : void
```

Les étages sont définis par une couleur et un numéro. A la création du système (**SystemInit**), les étages sont créés à travers notre propre liste chaînée. Ainsi nous pouvons simplement accéder à l'étage précédent ou suivant par la méthode getNextFloor() et getPreviousFloor(). Les méthodes goUp() et goDown() utilisent ces méthodes pour changer la position d'un **Elevator**. Les exceptions **LastFloorException** et **FirstFloorException** sont lancées si on essaie d'accéder respectivement au suivant du dernier **Floor** et au précédent du premier **Floor**.

Faire sortir les
utilisateurs à
leurs étages

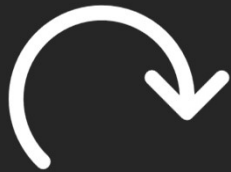


```
a
Elevator
<<Property>> -color : String
<<Property>> -maxWeight : int
-currentWeight : int = 0
<<Property>> #passengers : LinkedHashMap<User, Floor>
<<Property>> #reachableFloors : LinkedHashMap<Floor, Integer>
<<Property>> -direction : String
<<Property>> #elevatorNumber : int
-nbFloors : int
<<Property>> -nbDemandsTreated : int
<<Property>> -position : Floor

+Elevator(color : String, maxWeight : int, elevatorNumber : int, reachableFloors : LinkedHashMap<Floor, Integer>)
-weightCheck(u : User) : boolean
+enter() : void
+floorToElevator(pq : PriorityQueue<User>) : void
+addPassengersToElevator(pq : PriorityQueue<User>) : void
+exitWhenPMR() : void
+exit() : void
+goUp() : void
+goDown() : void
+getNbFloors() : int
+setNbFloors(nb : int) : void
+removeWeight(w : float) : void
+addNbDemandsTreated() : void
b
```

La méthode exit() permet de faire sortir les **Users** ayant pour destination la *position* de l'Elevator
La méthode removWeight() retire leur poids du *CurrentWeight* de l'Elevator

Séquence



Puisque nous sommes en programmation séquentielle, nous avons créé une classe **ElevatorSequence** pour gérer les séquences de notre système. La méthode `SystemEmpty` renvoie `true` quand le système n'a plus d'utilisateur en attente dans les **Floor** ni dans les **Elevator**. `makeSequence` fonctionne comme suit :

1. Appeler la fonction `dispatch()` pour attribuer les demandes

Pour chaque **Elevator** :

2. Gère les sorties
3. Gère les entrées
4. Gère les mouvements.

Ainsi, on peut pour faire fonctionner le système il suffit de coder :

^a	ElevatorSequence
	<u>+makeSequence() : void</u>
	<u>-elevatorStopper(el : Elevator) : void</u>
	<u>+SystemEmpty() : boolean</u>

```
SystemInit sys = new SystemInit();  
do {  
  
    ElevatorSequence.makeSequence();  
  
} while (!ElevatorSequence.SystemEmpty());
```

Resultats

```
INFORMATIONS USERS
*****
TOTAL USERS : 200.0
Number of PMR : 11.0
Proportion of PMR : 5.5%
Number of Students : 71.0
Proportion of Students : 35.5%
Number of Teacher : 59.0
Proportion of Teacher : 29.499998%
Number of Administrative : 59.0
Proportion of Administrative : 29.499998%
Medium Weight : 89.58015
Medium Age : 54.615

INFORMATIONS SYSTEM
*****
System Duration : 21ms
System Iterations : 24
System Success : 200 users
System failures : 0 users

INFORMATIONS ELEVATORS
*****
red
-----
1 - Number of demands treated : 2
2 - Number of demands treated : 3
3 - Number of demands treated : 2
4 - Number of demands treated : 2
5 - Number of demands treated : 3
6 - Number of demands treated : 2

Most demanded : Elevator n°2
green
-----
1 - Number of demands treated : 2
2 - Number of demands treated : 4
3 - Number of demands treated : 2
4 - Number of demands treated : 4
5 - Number of demands treated : 3
6 - Number of demands treated : 2
```

Floors

```
0, red
users up : 0
users down : 0

9, yellow
users up : 0
users down : 0

0, green
users up : 0
users down : 0

11, yellow
users up : 0
users down : 0

12, yellow
users up : 0
users down : 0

13, yellow
users up : 0
users down : 0

14, yellow
users up : 0
users down : 0

15, yellow
users up : 0
users down : 2

9, red
users up : 0
users down : 0
```

Users

```
Direction = down
Destination = 9, yellow
Final destination = 18, red
Source = 15, yellow
Is PMR : false

Direction = down
Destination = 5, green
Source = 8, green
Is PMR : true

Direction = down
Destination = 16, red
Source = 19, red
Is PMR : false

Direction = up
Destination = 9, green
Final destination = 20, red
Source = 0, green
Is PMR : false

Direction = up
Destination = 9, green
Final destination = 13, yellow
Source = 7, green
Is PMR : false
```

Demands

```
down, 15, yellow
down, 8, green
down, 19, red
up, 0, green
up, 7, green
down, 18, red
down, 9, green
```

Elevator

```
red : up : 9 : {} : 3
red : up : 9 : {} : 2
red : null : 0 : {} : 0
red : null : 0 : {} : 0
red : null : 0 : {} : 0
red : null : 0 : {} : 0
green : up : 4 : {} : 3
green : up : 4 : {user.Student@5cf87a79=floor.Floor@b6391527} : 0
green : up : 4 : {} : 0
green : up : 4 : {} : 4
green : null : 0 : {} : 0
green : null : 0 : {} : 0
yellow : up : 9 : {} : 5
yellow : null : 0 : {} : 0
yellow : null : 0 : {} : 0
yellow : null : 0 : {} : 0
yellow : null : 0 : {} : 0
yellow : null : 0 : {} : 0
```

IHM

Les méthodes suivantes permettent d'afficher les informations relatives aux différents objets du système pour connaître leurs états

- `Utils.displayDemandsDetails();`
- `Utils.displayElevatorDetails();`
- `Utils.displayFloorsDetails();`
- `Utils.displayUsersDetails`

SystemStats



La création d'une classe SystemStats nous a permis d'avoir des indicateurs sur la performance du système.

```


INFORMATIONS USERS
*****
TOTAL USERS : 200.0
Number of PMR : 11.0
Proportion of PMR : 5.5%
Number of Students : 71.0
Proportion of Students : 35.5%
Number of Teacher : 59.0
Proportion of Teacher : 29.499998%
Number of Administrative : 59.0
Proportion of Administrative : 29.499998%
Medium Weight : 89.58015
Medium Age : 54.615

INFORMATIONS SYSTEM
*****
System Duration : 21ms
System Iterations : 24
System Success : 200 users
System failures : 0 users

INFORMATIONS ELEVATORS
*****
red
-----
1 - Number of demands treated : 2
2 - Number of demands treated : 3
3 - Number of demands treated : 2
4 - Number of demands treated : 2
5 - Number of demands treated : 3
6 - Number of demands treated : 2

Most demanded : Elevator n°2
green
-----
1 - Number of demands treated : 2
2 - Number of demands treated : 4
3 - Number of demands treated : 2
4 - Number of demands treated : 4
5 - Number of demands treated : 3
6 - Number of demands treated : 2
    
```

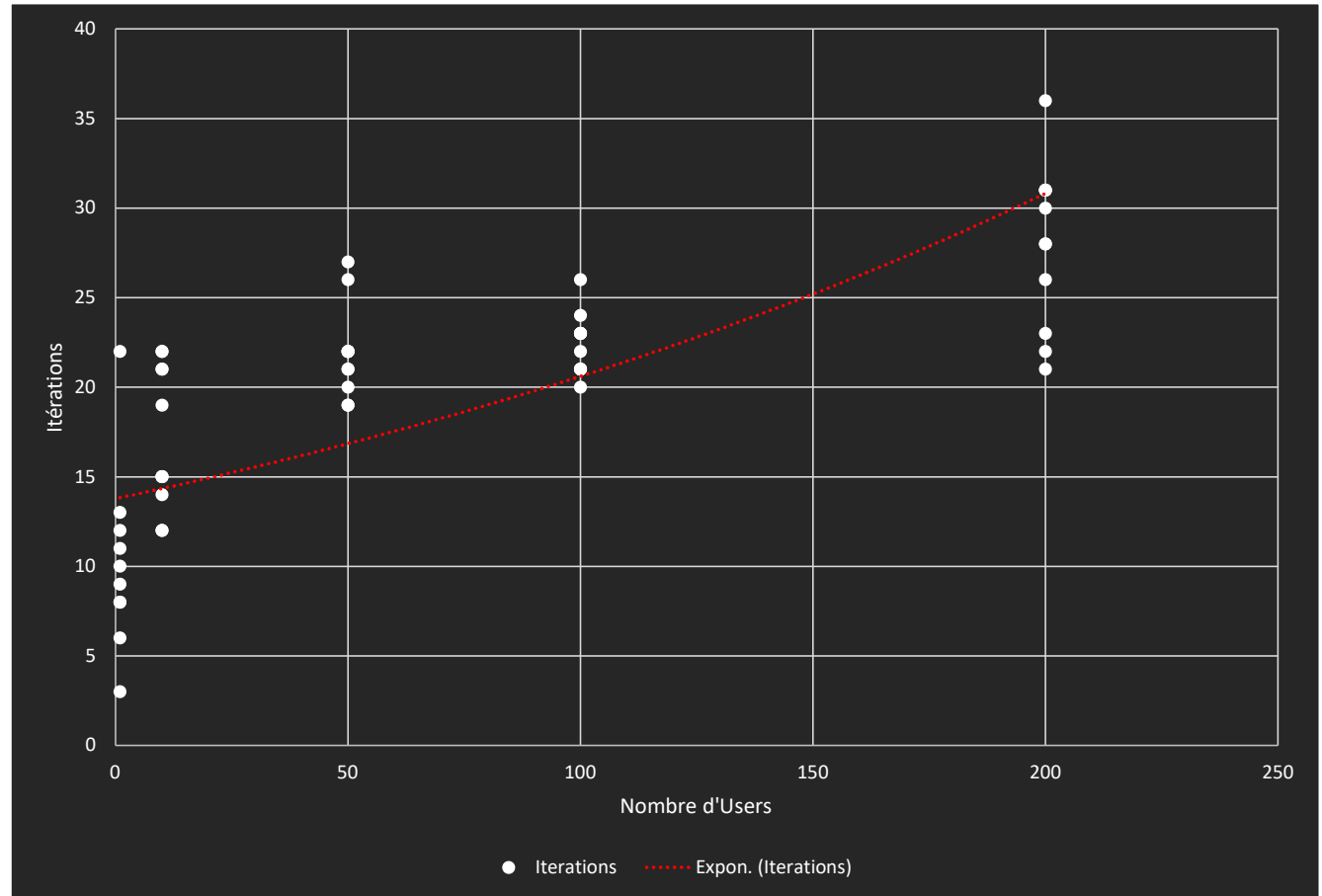
a SystemStats	
-nbSequenceliterations	: int
-totalUserReachDestination	: int
-timeStart	: long = 0
-timeEnd	: long = 0
-totalWeight	: float
-totalAge	: float
-nbPMR	: float
-nbStudent	: float
-nbTeacher	: float
-nbAdmin	: float
+getStats()	: String
+addPMR()	: void
+addAdmin()	: void
+addStudent()	: void
+addTeacher()	: void
+addAge(a : float)	: void
+addWeight(w : float)	: void
+addSequenceliteration()	: void
+setTimeStart()	: void
+setTimeEnd()	: void
+addUserReachDestination()	: void



Statistiques sur notre système

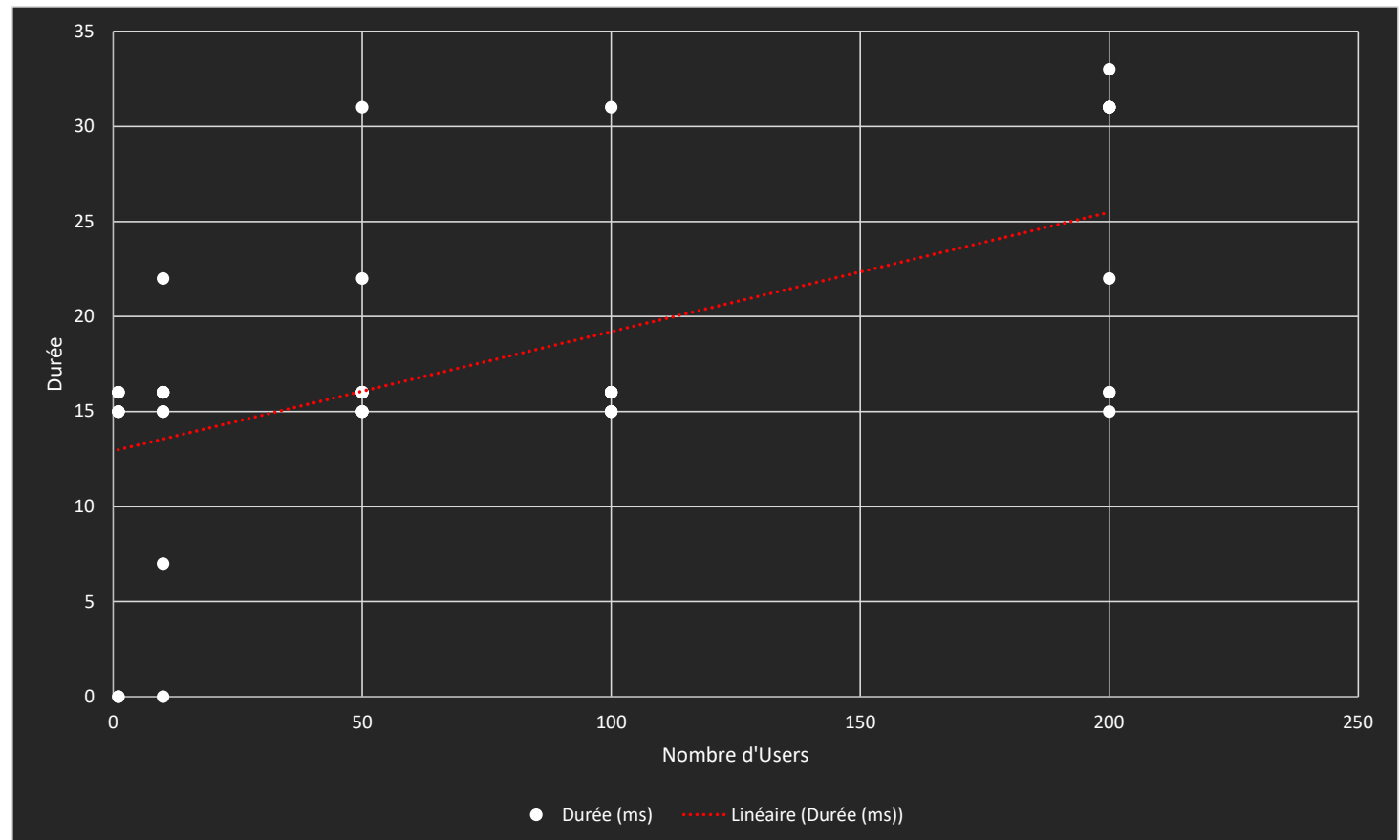
Corrélation
entre le
nombre
d'utilisateur et
le nombre
d'itérations

Coefficient de Pearson : 0,75480499



Corrélation entre
le nombre
d'utilisateur et la
durée
d'exécution du
programme

Coefficient de Pearson : 0,620707565



Corrélation
entre la durée
d'exécution du
programme et
le nombre
d'itérations

Coefficient de Pearson : 0,75480499

