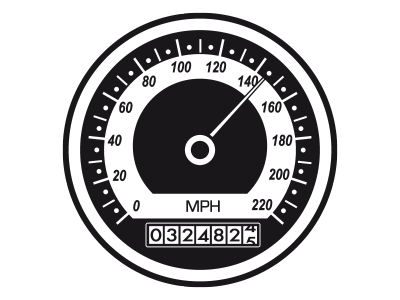
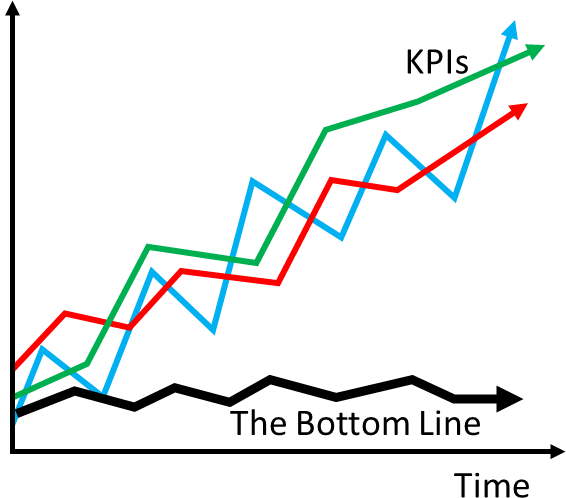
**LOGICIEL SIMERGY**

**RAPPORT**

[](https://www.google.fr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiGsba7q_HXAhVFrxoKHfKLCnwQjRwIBw&url=http%3A%2F%2Fwww.phone-service.fr%2Fqualite%2Fbarometre-qualite%2F&psig=AOvVaw1vnIeUICfaCRh91ujCr9ZY&ust=1512510662094173)





[](https://www.google.fr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiO09Trq_HXAhXEBBoKHebECUYQjRwIBw&url=http%3A%2F%2Fwww.frandroid.com%2Fandroid%2F319602_3d-touch-android-a-laide-barometre&psig=AOvVaw1vnIeUICfaCRh91ujCr9ZY&ust=1512510662094173)[](https://www.google.fr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjN_tKdrPHXAhXF2hoKHXJMAWIQjRwIBw&url=http%3A%2F%2Fwww.allaboutlean.com%2Fkpi-lies-examples%2Fkpi-vs-bottom-line-example%2F&psig=AOvVaw0x5CxCvdNwL0XVH-PmzguT&ust=1512510817591124)

**BRIEUC LOUSSOUARN LUDOVIC DESCATEAUX**

# Introduction

Le logiciel SimErgy est un outil de simulation conçu pour répondre au besoin de dimensionnement des départements hospitaliers d’urgence.

Il permet de simuler le processus de traitement d’un flux de patients à partir du paramétrage suivant :

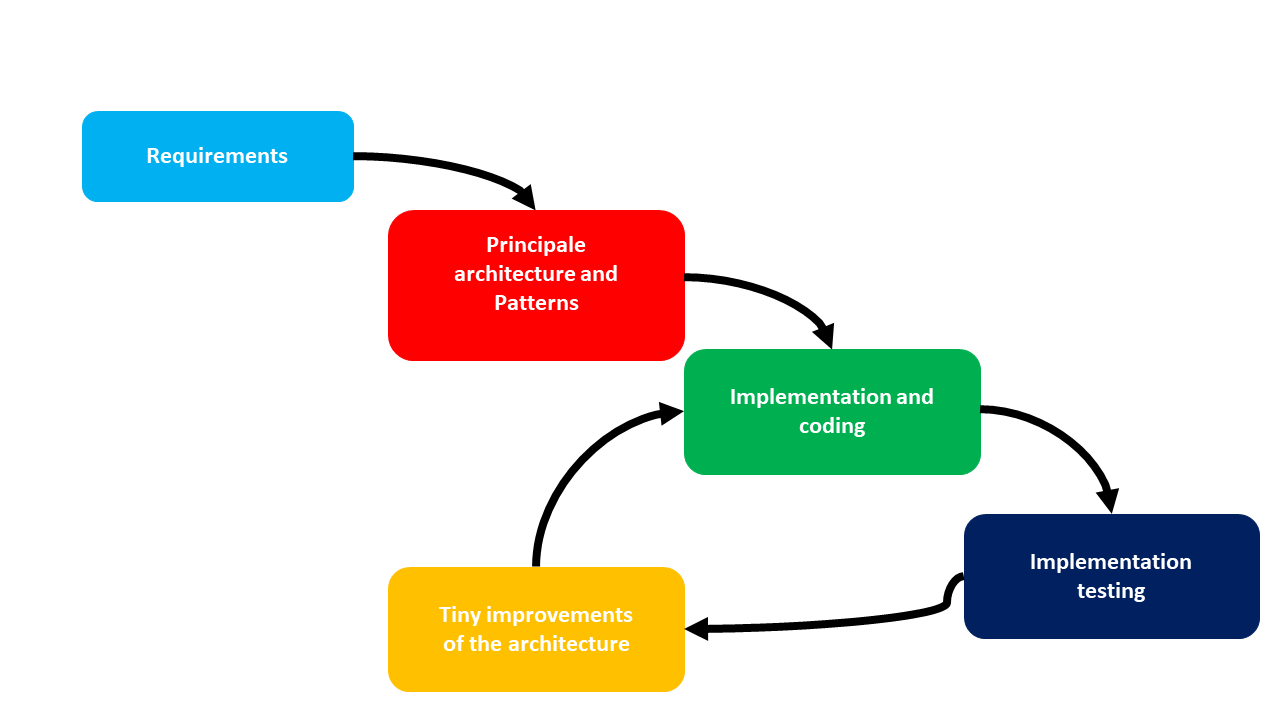
* Le flux de patient :
  + Typologie des patients : les patients n’ont pas tous le même degré de sévérité. Il est compris dans {‘L1’ (resuscitation) > ‘L2’ (emergency) > ‘L3’ (urgent) > ‘L4’ (less-urgent) > ‘L5’ (non-urgent)}.
  + Soit déterministe : on enregistre la typologie des patients et leur date d’arrivée avant de simuler le processus du département d’urgence.
  + Soit probabiliste : une distribution des arrivées sur un certain intervalle de temps est générée selon la loi de probabilité sélectionnée.
* Ressources humaines disponibles
  + Les Physicians qui diagnostiquent les patients et préconise un examen spécialisé si nécessaire (Radio, Blood, MRI).
  + Les Nurses qui peuvent enregistrer l’arrivée d’un patient, et le conduire en salle d’attente avant consultation d’un Physician.
  + Les Transporters qui transportent les patients entre leur salle de consultation et les salles de test spécialisé.
* Matériels mobiles :
  + Les Stretchers nécessaires au transport des patients graves ({‘L1’, ‘L2’}).
* Infrastructures :
  + Les WaitingRooms qui sont les salles d’attentes avant consultation.
  + Les Corridors qui sont les couloirs de l’hôpital, et qui servent de lieu d’attente dans le cas où toutes les WaitingRooms sont pleines.
  + Les ShockRooms qui sont les salles de consultation réservées aux patients de type {‘L1’, ‘L2’}.
  + Les BoxRooms qui sont les autres salles de consultation où tous les types de patients sont acceptés.
  + Les RadioRooms qui sont les salles de test radiographie.
  + Les MRIRooms qui sont les salles de test d’IRM.
  + Les BloodRooms qui sont les salles de test sanguins.

A partir de ce paramétrage initial, SimErgy permet de simuler le traitement du flux de patients et d’obtenir en temps réel les indicateurs de performance suivants :

* Le Door-to-doctor-time : temps moyen entre l’arrivée des patients et leur consultation.
* La Length-of-Stay : temps moyen de séjour au département (arrivées 🡪 verdict final)

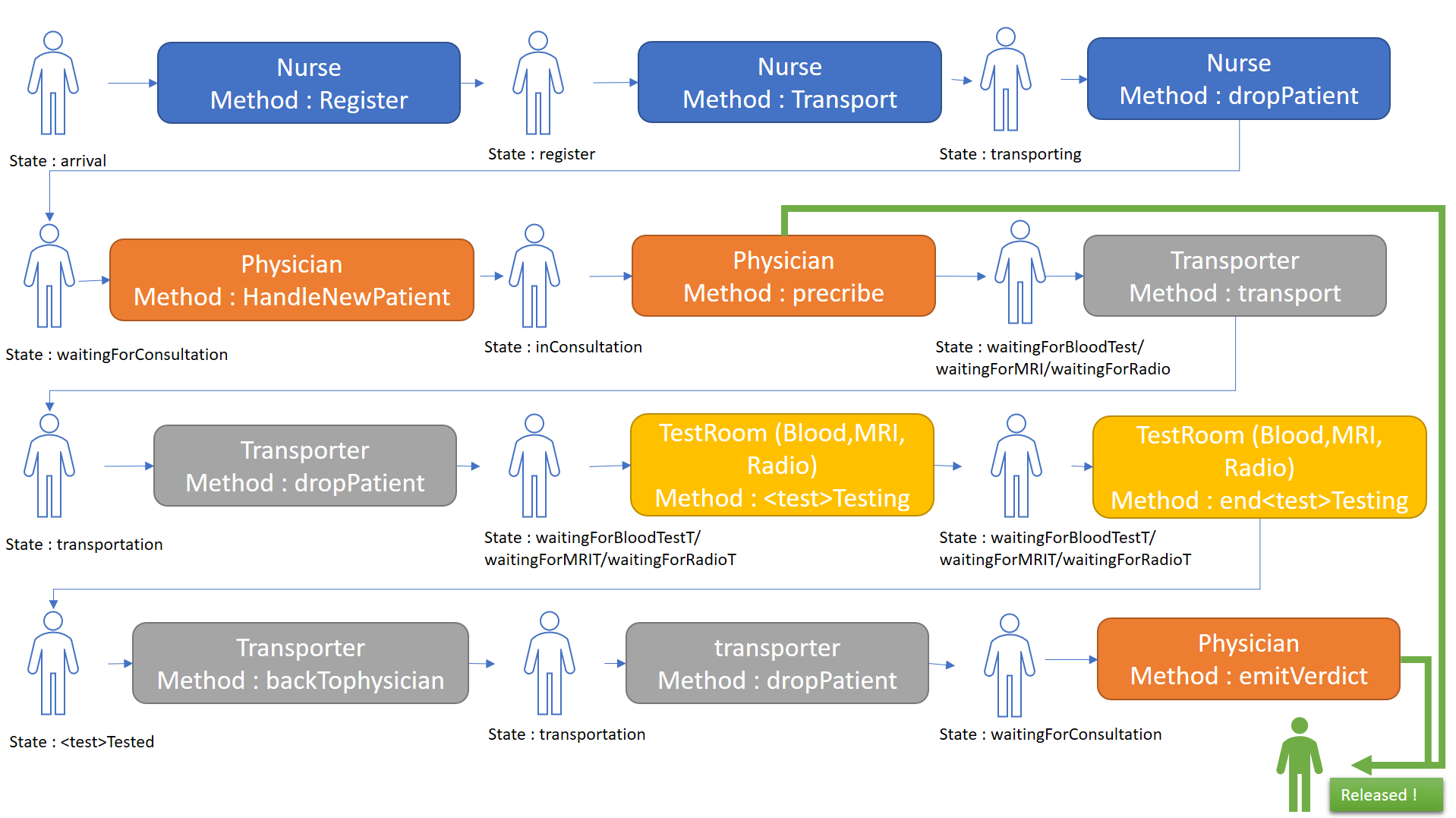
# Déroulement du projet de développement de SimErgy

Ce projet de réalisation du logiciel SimErgy a été mené en binôme : Brieuc LOUSSOUARN & Ludovic DESCATEAUX. Nous avons utilisé l’outil de version collaboratif Git afin de travailler en parallèle tout en conservant un travail commun cohérent. L’environnement de développement Papyrus nous a été d’une grande utilité notamment pour ses fonctions liant les diagrammes de classe au code, ses fonctions de génération de tests, et ses fonctions de débuggage pas à pas. Notre démarche de développement a été la suivante :

1. Prise en considération de toutes les exigences du cahier des charges
2. Elaboration de l’architecture-logiciel en intégrant des patterns visant une modularité afin d’envisager des extensions et mises à jour futures.
3. Implémentation et code
4. **Test de l’implémentation
5. Révision fine de l’architecture-logiciel

*Development pattern*

# Parcours d’un patient dans notre ED :

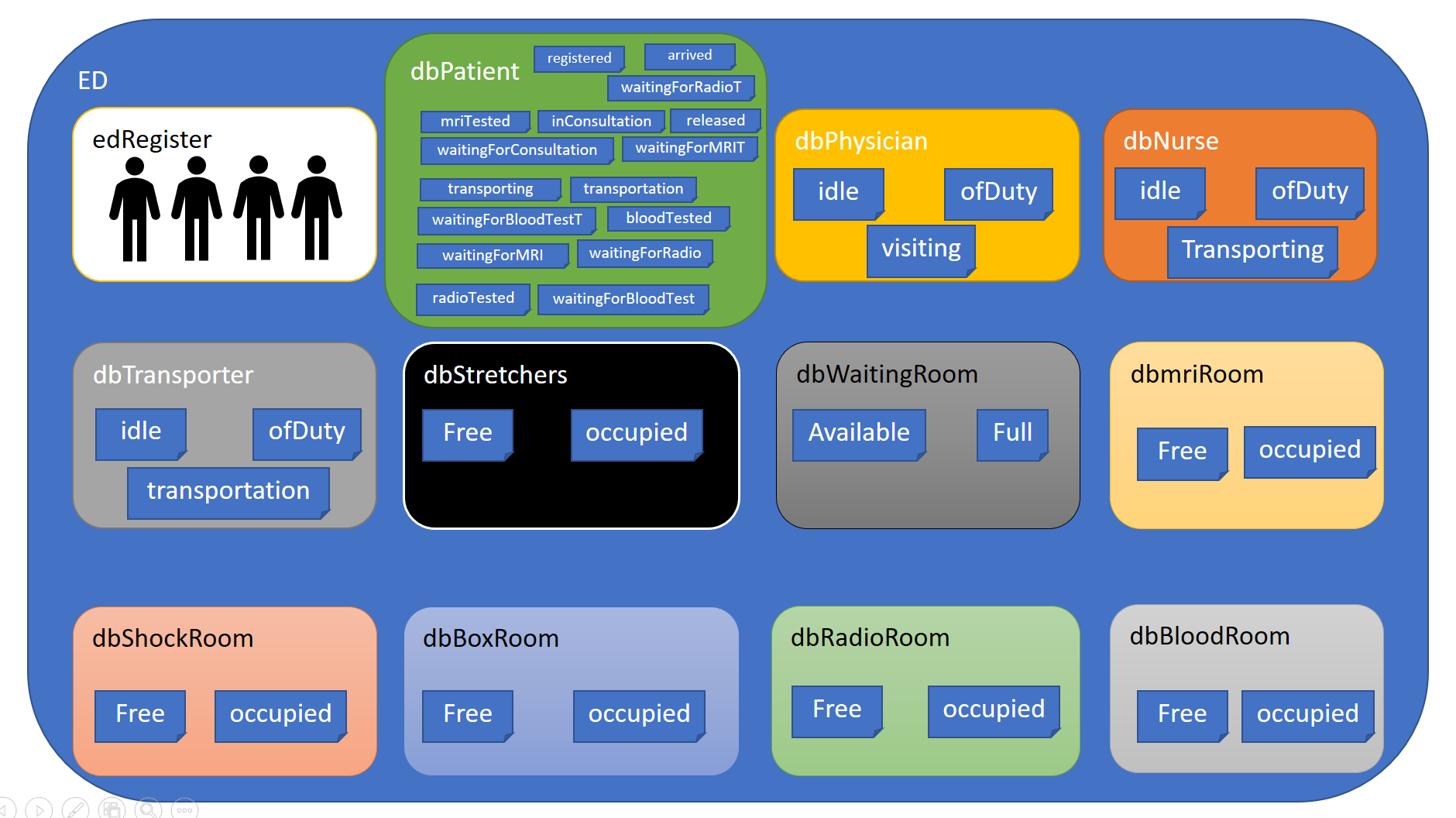


*Figure 1 : schéma de parcours d’un patient dans l’ED avec ses différents états*

Le patient est l’élément clef de notre simulateur. C’est lui qui subit la plupart des actions et dont on veut connaître l’état pour savoir si l’ED est bien performant. La figure 1 montre le parcours d’un patient de son arrivée à sa sortie avec les différents états qu’il peut adopter. Chaque modification de son état se fait avec une modification de sa place dans la base de données de l’ED (voir partie 3).

On peut noter que le physician traite le patient à chaque fois quand il est dans l’état waitingForConsultation. Cependant un fois il applique handleNewPatient et une autre fois emitVerdict. En effet, le physician est capable de savoir quoi faire car **il possède la liste des patients qu’il a vu**. Si il ne l’a jamais vu alors il doit appliquer handleNewPatient, sinon il doit appliquer emitVerdict.

# La gestion des états



*Figure 2 : base de données présente dans chaque classe ED*

Afin d’avoir une gestion des états de tous les éléments de chaque ED nous avons décidé d’équiper cette classe d’ArrayList. Elles sont toutes représentées dans le diagramme ci-dessus. Chaque case représente une liste et chaque étiquette bleue représente une sous liste avec l’état dédié. Leurs noms sont identiques à ceux du diagramme.

Une liste particulière est la liste edRegister. Elle comprend tous les patients qui ont été enregistré dans l’hôpital et cela quelque soit leur état.

Comment une entité (room, patient, ressource humaine…) passe d’une liste à l’autre lors de son changement d’état ? Cela est géré par les méthodes setState de chaque objet. Comme vous pouvez le voir dans le code, ces méthodes prennent comme argument une String avec l’état et se charge de modifier l’état et la place dans la database. Cela permet d’éviter les erreurs et de gérer parfaitement ces listes.

# Factory pattern and singleton pattern

Factory Pattern

Singleton Pattern

Un *singleton* *pattern* estappliqué au temps. En effet on veut avoir une instance du temps qui évolue pour tout le monde à chaque fois qu’elle est appelée. C’est pourquoi nous avons fait une classe temps et une classe *TimeStamp*. Les *TimeStamp* sont des images du temps à un instant donné. Ils disposent des méthodes permettant de renvoyer le temps sous une chaîne de caractère avec jour, heure et minute ou d’un nombre entier de minutes. Il faut donc voir que seul la méthode *timeGoes*() de l’unique instance de *Time* permet de faire avancer le temps.

Ainsi on est sûr que le temps évolue bien sur le simulateur.

# La gestion des messages

Les messages sont des outils pour signaler au médecin le statut de patients. Ils peuvent être envoyé par n’importe qui à n’importe quel moment. Ils peuvent être envoyé par tous le monde à n’importe quel médecin.

Un message a trois statuts : non-lu, lu ou supprimé. Pour modéliser cela nous avons équipé chaque médecin d’une *mailBox* qui est un *Arraylist* *d’Arraylist* de Message. *MailBox* est donc composé d’une liste comportant les messages non lus et d’une liste comportant les messages lus.

Afin d’effectuer les actions sur les messages et pouvoir modifier simplement ces actions les méthodes, *read*(), *unread*() et *remove*() sont codées dans la classe message. Nous avons ensuite codé des méthodes propres au *physician* qui font appels aux méthodes de *Message*. Il nous reste encore à ajouter une méthode *writeMessage*() à toutes les entités de l’ED.

Cela fonctionne bien car la JUnit vérifie bien toutes les contraintes et celles-ci est validée.

Le point faible de cette implémentation est que sans GUI on peut difficilement voir les messages et donc les appeler dans les méthodes. Cependant le codage de nos messages nous permet d’envisager un GUI qui afficherai les messages lu et non lu afin de les sélectionner.

# Calcul des KPI

Comme le demandait l’énoncé nous avons codé deux *KPIs* : LOS et DTDT. Nous avons pris le parti de faire deux classes qui héritent de la classe abstraite *Kpi* car elles ont toutes les deux besoin de deux méthodes : *toString*() et *calculate*().

Comme le veut aussi l’énoncé elles prennent en paramètre un ED et une sévérité de patient. En effet on veut savoir en fonction de la gravité le temps en « door to door » ou le temps de l’entrée à la sortie. Les Junit des KPIs vérifient ces propriétés et qui fonctionnent.