TD6 - Conception des Tests de conformité

Le test logiciel est la méthode la plus populaire pour vérifier un logiciel. Cette méthode représente environ **40-60% du prix final**. Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts

Plus précisément : une **spécification** est un ensemble explicite d'exigences à satisfaire par un logiciel, API, module, ….
Le **test** de conformité est l'exécution ou l'évaluation d'un système ou d'une composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.

Deux familles de tests utilisées simultanément dans l'industrie, car elles sont complémentaires :
Tests structurels (boîte de verre) : les donnés de tests sont produites après une analyse du code ; ces tests détectent principalement les erreurs commises.
Tests fonctionnels (boîte noire) : test de conformité par rapport à la spécification. Ces tests détectent principalement les erreurs d'omission par rapport à la spécification.

Le test est une activité créatrice:

- imaginer des scenarios plausibles pouvant mettre un logiciel en défaut
- imaginer et construire des bancs de tests permettant de vérifier les fonctionnalités et le respect des contraintes ...

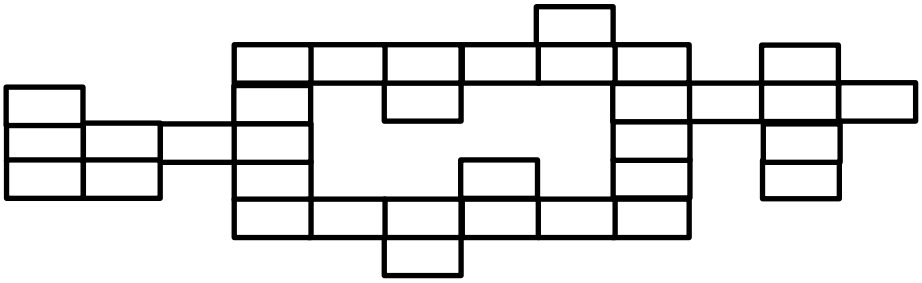
Quelques principes de base pour réaliser des tests.

- Un.e programmeur.e ne doit pas concevoir/écrire les bancs de tests ses propres programmes (tout simplement pour ne pas être juge et partie).
- Ne pas effectuer des tests avec l'hypothèse qu'aucune erreur ne va être trouvée.
- La définition des sorties ou résultats attendus doit être effectuée avant l'exécution d'un test ; car il y a des chances non négligeable de prendre un résultat erroné mais semblant cohérent pour un résultat correct.
- Inspecter minutieusement les résultats (les traces) de chaque test.
- Les jeux de tests doivent être écrits pour des entrées invalides ou incohérentes.

L'objectif de ce TD est d'apprendre à concevoir des jeux de tests de conformité pertinents.

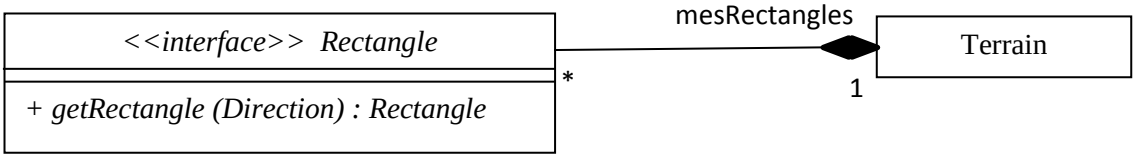
Conception d'un jeu de robots sur un terrain

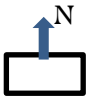
Info 1: Terrain irrégulier composé de rectangles de même taille, pouvant comporter des « vides ». L'appel *rec.getRectangle(dir)* doit retourner le Rectangle voisin de l'objet *rec* dans la direction *dir*, si ce rectangle existe dans le terrain ; sinon *null* est retourné.



Hypothèse : les méthodes permettant de construire un terrain ont déjà été testées.

Ebauche du diagramme de classes :



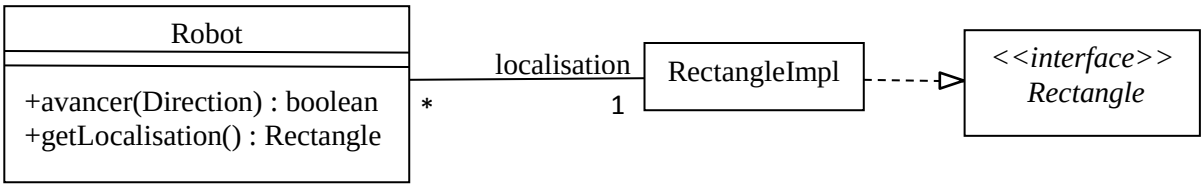



```
public enum Direction { NORD, EST,  SUD,  OUEST; }
```

Question 1 Proposez un terrain et des tests pour l'implémentation de *getRectangle(Direction).*

Info 2: un robot est sur un rectangle, il peut se déplacer dans les 4 directions sur le terrain.

Ebauche 2 du diagramme de classes :



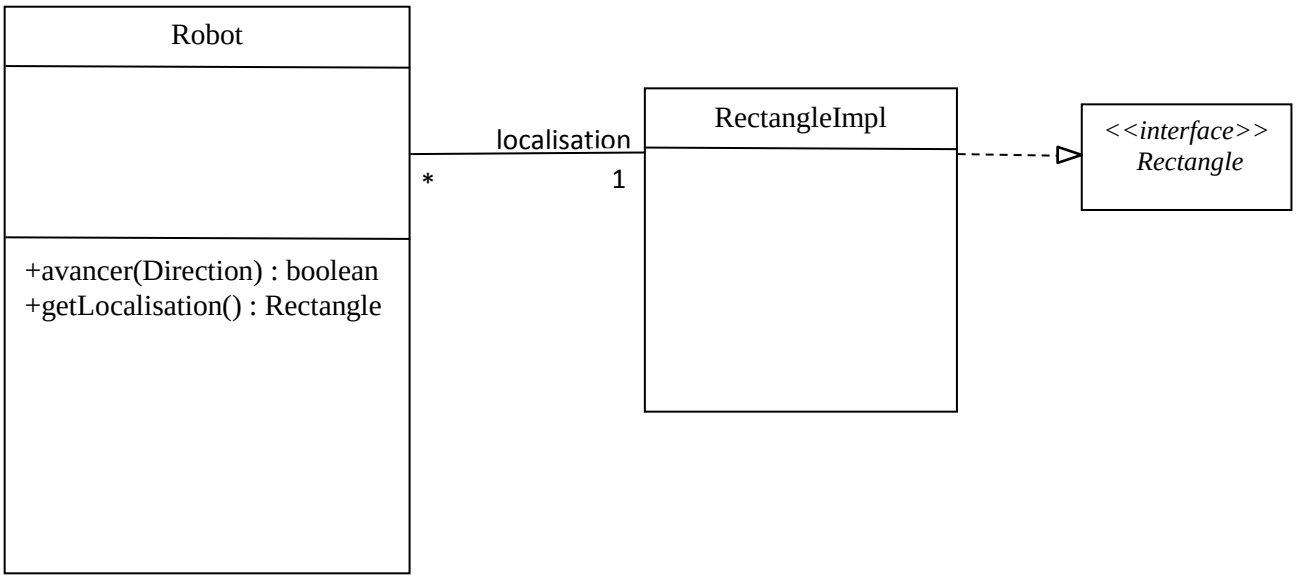
L'appel *rob.avancer(dir)* avance le robot *rob* dans la direction indique dans *dir* si cela est possible et retourne *true* sinon l'appel retourne *false*.

Question 2 Proposez un terrain et des tests pour l'implémentation de la méthode *avancer(Direction).*

Info 3: certaines cases contiennent un trésor caché (des pièces d'argent) ; le premier robot qui passe sur la case gagne le trésor.

Questions 3 Complétez l'ébauche 3 du diagramme de classes.
Proposez un autre terrain et des tests complémentaires pour l'implémentation de la méthode *avancer(Direction)* en fonction de l'info 3.

Ebauche 3 du diagramme de classes :



Info 4 : Lorsqu'un robot arrive sur un rectangle contenant un autre robot, le plus fort vole 10% du trésor du plus faible

Questions 4 Complétez l'ébauche 3 du diagramme de classes.
Proposez un quatrième terrain et des tests supplémentaires pour l'implémentation de la méthode *avancer(Direction)* en fonction de l'info 4.

Info 5: un robot peut se renforcer (augmenter sa force) mais cela a un coût 100 pièces pour une augmenter la force de 1. Ce montant est prélevé sur la fortune du robot. Si le robot n'est pas assez riche, le rebot ne peut pas augmenter sa force.

Questions 5 Complétez l'ébauche 3 du diagramme de classes.
Proposez un quatrième terrain et des tests pour l'implémentation de la méthode *renforcer(int)* en fonction de l'info 5.