

# Conception et Programmation orientées objets avancées

## Le design pattern *Stratégie*

---

### Problème

On souhaite gérer des robots afin qu'ils déplacent des objets présents dans un casier contenant des cases numérotées.

### UML

- 1) Un robot est toujours associé à un unique casier, et plusieurs robots peuvent manipuler un même casier. Faire un diagramme de classes modélisant cette situation.
- 2) Tout robot possède un déplacement lui permettant d'aller d'une case à l'autre. On connaît deux façons de se déplacer : *voler* ou *rouler* qui s'appliquent à une opération *seDeplacer()* mettant en œuvre un seul de ces déplacements. Proposer une conception objet qui encapsule les comportements de déplacement. Compléter le diagramme de classes en conséquence.
- 3) Modifier le diagramme pour qu'un *Robot* puisse exécuter le type de déplacement qui lui est associé.
- 4) Tout robot possède une capacité de manipulation des objets : soit par magnétisme soit par pincement. Le type de manipulation d'un robot (*magnétiser* ou *pincer*) s'appliquent aux opérations *saisir()* et *relacher()*. Proposer une conception objet qui encapsule les comportements de manipulation. Compléter le diagramme de classes en conséquence.
- 5) Modifier le diagramme pour qu'un *Robot* puisse exécuter le type de saisie et de dépose d'objets qui lui est associé.
- 6) Compléter le diagramme de classes en ajouter les classes *Drone* et *AutoTracteur*, qui héritent de *Robot*.

### java

Consulter la documentation des classes *Casier* et *Objet* fournie en annexe.

- 1) Écrire l'interface et les classes encapsulant les comportements de déplacement. La méthode *seDeplacer()* affichera «*je roule*» ou «*je vole*».
- 2) Écrire l'interface et les classes encapsulant les comportements de manipulation. La méthode *saisir()* affichera «*je magnétise*» ou «*je ferme la pince* », la méthode *lacher()* affichera «*je démagnétise*» ou «*j'ouvre la pince*».

### 3) Écrire une classe *Robot*

déclarer :

- une variable *int numeroCase* représentant le numéro de la case devant laquelle est le robot,
- une variable *Objet monObjet* représentant l'objet que tient le robot,
- une variable *Casier monCasier*,
- une variable représentant le comportement de déplacement,
- dune variable représentant le comportement de manipulation.

écrire :

- un constructeur *Robot(Casier unCasier)* qui affecte *unCasier* au robot, et initialise les variables *monObjet* à *null* et *numeroCase* à 0 (le robot ne tient pas d'objet et se situe devant la première case du casier).
  - une méthode *executerSeDeplacer(int numeroCaseArrivee)* qui exécute le déplacement du robot vers une nouvelle case. Afficher la case de départ, le nature du déplacement effectué et la case d'arrivée.
  - Une méthode *executerPrendre()* qui affiche le type de prise et l'objet saisi dans la case devant laquelle se situe le robot. A l'issue de la prise le robot tient l'objet saisi qui n'est plus dans la case.
  - Une méthode *executerLacher()* qui affiche le type de dépose et l'objet déposé dans la case devant laquelle se situe le robot. A l'issue de la prise le robot ne tient plus l'objet qui apparaît dans la case.
- 4) Écrire une classe *Drone* représentant un robot qui se déplace en volant et magnétise les objets qu'il manipule.
- 5) Écrire une classe *AutoTracteur* représentant un robot qui se déplace en roulant et pince les objets qu'il manipule.
- 6) Écrire une classe *Simulation\_robots* qui :
- créé un *Casier* de 3 cases contenant un unique objet à la troisième case.
  - créé une *Drone* qui déplace l'objet de la troisième case à la seconde (afficher l'état initial et final du casier)
  - créé une *AutoTracteur* qui déplace l'objet de la seconde case à la première (afficher l'état initial et final du casier)
- 7) (question supplémentaire) ajouter à la classe *Robot* les exceptions nécessaires pour gérer les situations conflictuelles

### TP

- Récupérer les classes *Casier* et *Objet*
- Faire un projet *Éclipse* contenant l'ensemble des classes, tester les classes en utilisant le debugger.

## ANNEXE

### public class Casier

Constructeur(s)	
<u>Casier</u> (int n) initialise le <i>Casier</i> en créant <i>n</i> cases	
Méthodes	
Objet	<u>libere objet</u> (int i) retourne l' <i>Objet</i> présent a la case <i>i</i> du <i>Casier</i> (l' <i>Objet</i> est retire de la case qui contient <i>null</i> après l'opération)
int	<u>nbCases</u> ( ) retourne le nombre de cases du <i>Casier</i>
void	<u>prend objet</u> (Objet e, int i) dépose l' <i>Objet</i> e dans la case <i>i</i> du <i>Casier</i>
String	<u>toString</u> ( ) retourne la description de tous les <i>Objets</i> du <i>Casier</i>

### public class Objet

Constructeur(s)	
<u>Objet</u> (String unNom, int unPoids) initialise l' <i>Objet</i> avec <i>unNom</i> et <i>unPoids</i>	
Méthodes	
String	<u>getNom</u> ( ) retourne le nom de l' <i>Objet</i>
int	<u>getPoids</u> ( ) retourne le poids de l' <i>Objet</i>
String	<u>toString</u> ( ) retourne la description de l' <i>Objet</i>

### Références pour l'utilisation Eclipse

- <http://www.eclipsetotale.com/articles/premierPas.html>
- <http://jmdoudoux.developpez.com/cours/developpons/eclipse/>
- <http://eclipse.developpez.com/faq/>