IUT Villetaneuse
M3105 : Conception et programmation objet avancées
Année 2019/2020, Formation Initiale

TD2 : Le patron de conception Stratégie

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

I 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 prendre() et lacher(). 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.
Correction :

◇

II Java

Consulter la documentation des classes Casier et Objet fournie en annexe. Récupérer les classes Casier et Objet [1]. Faire un projet Éclipse contenant l'ensemble des classes, tester les classes en utilisant le debugger.

1. Les sources sont accessibles à `https://www.lipn.univ-paris13.fr/~zargayouna/S3-M3105/sources-strategie/`

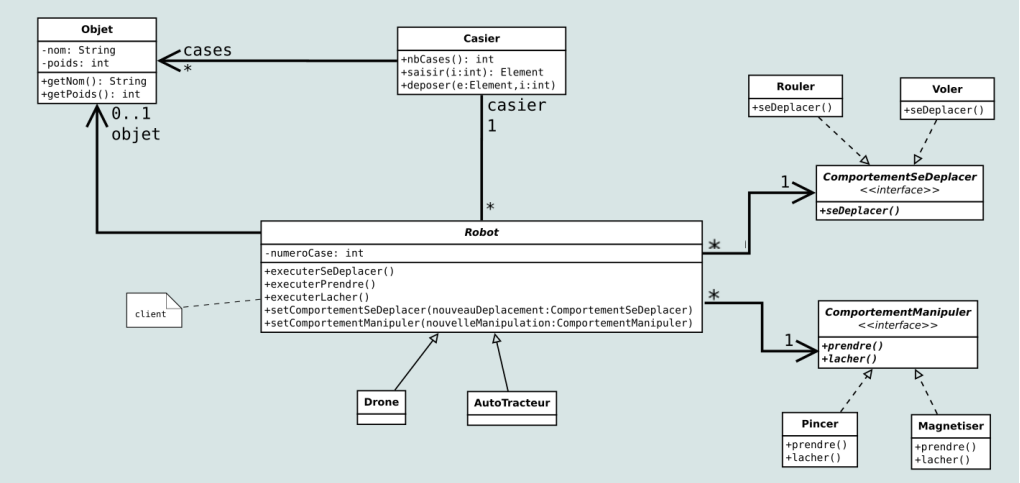

FIGURE 1 – Diagramme complet

1° Écrire l'interface et les classes encapsulant les comportements de déplacement. La méthode seDeplacer() affichera «je roule» ou «je vole».
Correction :

```
public interface ComportementSeDeplacer
{

public abstract void seDeplacer() ;

}

public class Rouler implements ComportementSeDeplacer
{
public void seDeplacer()
{
System.out.print("je roule") ;
}
}

public class Voler implements ComportementSeDeplacer
{
public void seDeplacer()
{
System.out.print("je vole") ;
}
}
```

◇

2° Écrire l'interface et les classes encapsulant les comportements de manipulation. La méthode prendre() affichera «je magnétise et je prends » ou « je ferme la pince et je prends », la méthode lacher() affichera «je démagnétise et je dépose» ou «j'ouvre ma pince et je dépose ».
Correction :

```
public interface ComportementManipuler
```

```
{

public abstract void prendre() ;

public abstract void lacher()  ;

}
public class Pincer implements ComportementManipuler
{

public void prendre()
{
System.out.print("je ferme la pince et je prends ") ;
}

public void lacher()
{
System.out.print("j'ouvre ma pince et je dépose ") ;
}

}
// fin classe Pincer

public class Magnetiser implements ComportementManipuler
{

public void prendre()
{
System.out.print("je magnetise et je prends ") ;
}

public void lacher()
{
System.out.print("je démagnétise et je depose ") ;
}

}
// fin classe Magnetiser
```

◇

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.

Correction :

```
*/
public abstract class Robot
{
private Casier casier ;// Casier que peut explorer le Robot
private int numeroCase ;// numero de case devant lequel se situe le Robot
private Objet objet ;// objet que tient le Robot (null s'il ne tient rien)

private ComportementSeDeplacer unComportementSeDeplacer ;
private ComportementManipuler unComportementManipuler ;


/**
 *
 */
public Robot(Casier unCasier)
{
this.objet = null ;
this.numeroCase = 0 ;
this.casier = unCasier ;
}

/**
 * retourne le comportementDeplacement du Robot courant
 */
public ComportementSeDeplacer getComportementSeDeplacer()
{
return this.unComportementSeDeplacer ;
}


/**
 * retourne le comportementManipuler du Robot courant
 */
public ComportementManipuler getComportementManipuler()
{
return this.unComportementManipuler ;
}


/**
 * retourne l'Objet que tient le robot (null s'il n'a aucun objet)
```

```
 */
public Objet getObjet()
{
return this.objet ;
}


/**
 * retourne le numero de case devant laquelle se tient le robot
 */
public int getNumeroCase()
{
return this.numeroCase ;
}


/**
 * affecte nouvelleCase comme nodu Robot
 */
public void setNumeroCase(int nouvelleCase)
{
this.numeroCase = nouvelleCase ;
}


/**
 * affecte nouvelObjet comme Objet du Robot
 */
public void setObjet(Objet nouvelObjet)
{
this.objet = nouvelObjet ;
}


/**
 *
 */
public void setComportementManipuler (ComportementManipuler
nouvelleFaconManipuler)
{
this.unComportementManipuler = nouvelleFaconManipuler ;
}


/**
 *
 */
public void setComportementSeDeplacer(ComportementSeDeplacer
nouveauDeplacement)
{
this.unComportementSeDeplacer = nouveauDeplacement ;
}
```

```
/**
 * execute le deplacement du Robot courant
 */
public void executerSeDeplacer(int arrivee)
{
System.out.println(this) ;

this.unComportementSeDeplacer.seDeplacer();


this.setNumeroCase(arrivee) ;
System.out.print(" vers la case " + this.getNumeroCase()) ;
System.out.println("\n") ;
}


/**
 * execute l'action prendre du Robot courant et retourne l'Objet pris
 */
public void executerPrendre()
{
System.out.println(this) ;
this.unComportementManipuler.prendre();

this.objet = this.casier.libere_objet(this.getNumeroCase()) ;

// possible this.setObjet(this.casier.saisir(this.getNumeroCase())) ;

System.out.println(" un(e) " + this.getObjet() + " dans la case " +
 this.getNumeroCase()) ;
System.out.println("\n") ;
}

/**
 * execute l'action lacher du Robot courant
 */
public void executerLacher()
{
System.out.println(this) ;
this.unComportementManipuler.lacher();

this.casier.prend_objet(this.getObjet(), this.getNumeroCase()) ;
// possible this.setObjet(this.casier.saisir(this.getNumeroCase())) ;

System.out.println("un(e) " + this.getObjet() + " dans la case " +
 this.getNumeroCase()) ;
System.out.println("\n") ;
}

/**
 * retourne le String decrivant le Robot courant
 */
public String toString()
```

```
{
String s = "je suis a la case " + this.getNumeroCase() ;

if (this.getObjet() == null)
s = s + " et je ne tiens rien" ;
else
s = s + " et je tiens un(e) " + this.getObjet() ;

return s ;
}

}
// fin classe Robot
```

◇

4° Écrire une classe Drone représentant un robot qui se déplace en volant et magnétise les objets qu'il manipule.
Correction :

```
public class Drone extends Robot {

/**
 * initialise le Drone courant en lui affectant une
 * saisie magnatique et un déplacement en volant
 */
public Drone(Casier unCasier)
{
super(unCasier) ;
this.setComportementSeDeplacer(new Voler()) ;
this.setComportementManipuler(new Magnetiser()) ;
}

public String toString()
{
String s = super.toString();

return "je suis un Drone, " + s ;
}
}
// fin Classe Drone
```

◇

5° Écrire une classe AutoTracteur représentant un robot qui se déplace en roulant et pince les objets qu'il manipule.
Correction :

```
public class Autotracteur extends Robot {
/**
 * initialise l'AutoTracteir courant en lui affectant une
 * saisie par pince et un déplacement en roulant
```

```
 */
public Autotracteur(Casier unCasier)
{
super(unCasier) ;
this.setComportementSeDeplacer(new Rouler()) ;
this.setComportementManipuler(new Pincer()) ;
}
/**
 *
 */
public String toString()
{
String s = super.toString();

return "je suis un AutoTracteur " + s ;
}
}
// fin Classe Autotracteur
```

◇

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)

Correction :

```
public class SimulationRobot {

public static void main(String[] args) {
/*
* 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)
 */
Casier c = new Casier(3) ;// initialisation d'un Casier
c.prend_objet(new Objet("guitare", 4),2) ;


System.out.println("etat initial du casier ") ;
System.out.println(c) ;


Robot r1 = new Drone(c) ;// initialisation d'un Robot
```

```
r1.executerSeDeplacer(2) ;
r1.executerPrendre()  ;
r1.executerSeDeplacer(1) ;
r1.executerLacher() ;

Robot r2 = new Autotracteur(c) ;

r2.executerSeDeplacer(1) ;
r2.executerPrendre()  ;
r2.executerSeDeplacer(0) ;
r2.executerLacher() ;

System.out.println("etat final du casier ") ;
System.out.println(c) ;
}

}
```

◇

7° (question supplémentaire) ajouter à la classe Robot les exceptions nécessaires pour gérer les situations conflictuelles

ANNEXE

Références pour l'utilisation Eclipse
http://www.eclipsetotale.com/articles/premierPas.html
http://jmdoudoux.developpez.com/cours/developpons/eclipse/
http://eclipse.developpez.com/faq/

public class Casier

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

public class Objet

Constructeur(s)	
`Objet(String unNom, int unPoids)` initialise l'*Objet* avec *unNom* et *unPoids*	
	Méthodes
`String`	`getNom()` retourne le nom de l'*Objet*
`int`	`getPoids()` retourne le poids de l'*Objet*
`String`	`toString()` retourne la description de l'*Objet*

FIGURE 2 – JavaDoc des classes Casier et Objet