



École Supérieure Polytechnique(ESP)  
Département Génie Informatique

# Rapport Mini Projet Final

## Les robots pollueurs

Module : JAVA\_POO

DIC1

Groupe 10

Réalisé par :

- Hamidou Woury Ba
- Cheikh Seck
- Jean Robert Sarr
- Marie Wally Fall
- Nana Katy Aidara

Sous la direction de :

- Mr Ibrahima Fall

Année 2022/2023

---

## Les classes et méthodes

Nous avons 8 classes à savoir :

1. La classe **Monde** qui est constituée de 6 méthodes :
  - (a) La méthode **toString** : cette méthode retourne une chaîne de caractères décrivant le monde.
  - (b) La méthode **estValide** : cette méthode permet de vérifier que la case demandée appartient à la matrice.
  - (c) La méthode **metPapierGras** : cette méthode met un papier gras dans la case (i; j).
  - (d) La méthode **prendPapierGras** : cette méthode enlève le papier gras de la case (i; j).
  - (e) La méthode **estSale** : cette méthode teste si la case (i, j) a un papier gras.
  - (f) La méthode **nbPapiersGras** : cette méthode rend le nombre de papier gras dans le monde.
2. La classe **Robot** qui est une classe abstraite, elle est constituée de 3 méthodes :
  - (a) La méthode **estValide** : cette méthode permet de vérifier que la case demandée appartient à la matrice.
  - (b) La méthode **vaEn** : cette méthode se déplace en (i, j).
  - (c) La méthode **parcourir** : cette méthode permet aux robots de parcourir le monde. Comme chaque robot a sa manière de parcourir le monde, on devra donc l'implémenter selon cette condition dans les sous-classes.
3. La classe **PollueurToutDroit** qui hérite de **Robotpollueur**, donc elle hérite des méthodes et attributs de la classe **RobotPollueur**. Puisqu'elle hérite d'une classe abstraite, nous avons redéfini la méthode abstraite **parcourir**.
4. La classe **Robotpollueur** en plus d'être une classe abstraite, elle hérite de la classe **Robot**, donc elle hérite de tous les attributs et de toutes les méthodes de sa classe mère. Elle possède également une autre méthode :
  - (a) La méthode **polluer** : cette méthode met un papier gras là où ce robot se trouve dans le monde.
5. La classe **PollueurSauteur** qui hérite de **Robotpollueur**, donc elle hérite des méthodes et attributs de la classe **RobotPollueur**. Puisque **PollueurSauteur** hérite d'une classe abstraite, nous avons redéfini la méthode abstraite **parcourir**.
6. La classe **RobotNettoyeur** qui hérite de **Robot**, donc elle hérite des méthodes et attributs de la classe **Robot**. Puisque **PollueurNettoyeur** hérite d'une classe abstraite, nous avons redéfini la méthode abstraite **parcourir**. Cette classe possède également une autre méthode :
  - (a) La méthode **nettoyer** : enlève le papier gras de la case où se trouve ce robot.
7. La classe **NettoyeurDistrait** qui hérite de **Robot**, donc elle hérite des méthodes et attributs de la classe **Robot**. Puisque **PollueurDistrait** hérite d'une classe abstraite, nous avons redéfini la méthode abstraite **parcourir**.
8. La classe **TestRobots** : cette classe contient un programme de test des classes précédentes.

**Remarque :** Chaque classe est constituée d'un ou de deux constructeurs sauf la classe TestRobots

---

## Code des classes

### Code de la classe Monde.java

```
1 package Mondepacage;  
2  
3 /**  
4  * Cette classe represente un objet Monde qui contient des informations  
5   * sur les cases et les robots.  
6  */  
7 public class Monde {  
8     public int nbL; // Le nombre de lignes de la matrice  
9     public int nbC; // Le nombre de colonnes de la matrice  
10    public boolean mat[][]; // La matrice booleenne  
11  
12    /**  
13     * Constructeur par defaut qui cree un monde 10x10 sans papiers gras.  
14     */  
15    public Monde() {  
16        this.nbL = 10;  
17        this.nbC = 10;  
18        this.mat = new boolean[nbL][nbC];  
19    }  
20  
21    /**  
22     * Constructeur qui cree un monde avec un nombre donne de lignes et de  
23     * colonnes sans papiers gras.  
24     *  
25     * @param nbL Le nombre de lignes du monde  
26     * @param nbC Le nombre de colonnes du monde  
27     */  
28    public Monde(int nbL, int nbC) {  
29        this.nbL = nbL;  
30        this.nbC = nbC;  
31        this.mat = new boolean[nbL][nbC];  
32    }  
33  
34    /**  
35     * Retourne une chaine de caracteres decrivant le monde.  
36     * Utilise "o" pour represente vrai et "." pour represente faux.  
37     *  
38     * @return La representation en chaine de caracteres du monde  
39     */  
40    public String toString() {  
41        StringBuilder chaine = new StringBuilder();  
42        for(int i = 0; i < nbL; i++){  
43            for(int j = 0; j < nbC; j++){  
44                if(mat[i][j])  
45                    chaine.append(" o ");  
46                else  
47                    chaine.append(" . ");  
48            }  
49            chaine.append("\n");  
50        }  
51        return chaine.toString();  
52    }  
53  
54    /**  
55     * Verifie si les coordonnees (i, j) appartiennent a la matrice.
```

```

54      *
55      * @param i La coordonnee i de la case
56      * @param j La coordonnee j de la case
57      * @return true si les coordonnees sont valides, sinon false
58      */
59      private boolean estValide(int i, int j) {
60          return i >= 0 && i < nbL && j >= 0 && j < nbC;
61      }
62
63      /**
64       * Place un papier gras dans la case specifiee par les coordonnees (i,
65       * j).
66       *
67       * @param i La coordonnee i de la case
68       * @param j La coordonnee j de la case
69       */
70      public void metPapierGras(int i, int j) {
71          if(estValide(i,j))
72              mat[i][j] = true;
73      }
74
75      /**
76       * Enleve le papier gras de la case specifiee par les coordonnees (i,
77       * j).
78       *
79       * @param i La coordonnee i de la case
80       * @param j La coordonnee j de la case
81       */
82      public void prendPapierGras(int i, int j) {
83          if(estValide(i,j))
84              mat[i][j] = false;
85      }
86
87      /**
88       * Verifie si la case specifiee par les coordonnees (i, j) a un papier
89       * gras.
90       *
91       * @param i La coordonnee i de la case
92       * @param j La coordonnee j de la case
93       * @return true si la case a un papier gras, sinon false
94       */
95      public boolean estSale(int i, int j) {
96          if(estValide(i,j))
97              return mat[i][j];
98          return false;
99      }
100
101      /**
102       * Renvoie le nombre de papiers gras presents dans le monde.
103       *
104       * @return Le nombre de papiers gras dans le monde
105       */
106      public int nbPapiersGras() {
107          int compteur = 0;
108          for(int i = 0; i < nbL; i++)
109              for(int j = 0; j < nbC; j++)
110                  if(mat[i][j])
111                      compteur++;
112
113          return compteur;
114      }
115  }

```

---

## Codes de la classe Robot.java

```
1 package Robotpackage;
2
3 import Mondepackage.Monde;
4
5 /**
6  * Cette classe represente un objet Robot abstrait.
7  * Un Robot a une position (posx, posy) dans le monde et peut se deplacer.
8  */
9 public abstract class Robot {
10     public int posx; // posx, posy : position du robot sur le monde.
11     public int posy;
12     public Monde m;
13
14     /**
15      * Constructeur qui cree un robot avec une position donnee dans le
16      * monde donne.
17      *
18      * @param x La coordonnee x de la position du robot
19      * @param y La coordonnee y de la position du robot
20      * @param m Le monde dans lequel le robot evolue
21      */
22     public Robot(int x, int y, Monde m) {
23         this.posx = x;
24         this.posy = y;
25         this.m = m;
26     }
27
28     /**
29      * Constructeur qui cree un robot avec une position aleatoire dans le
30      * monde donne.
31      *
32      * @param m Le monde dans lequel le robot evolue
33      */
34     public Robot(Monde m) {
35         this((int)(Math.random() * m.nbL), (int)(Math.random() * m.nbC),
36             m);
37     }
38
39     /**
40      * Verifie si la case aux coordonnees (i, j) appartient a la matrice
41      * du monde.
42      *
43      * @param i La coordonnee i
44      * @param j La coordonnee j
45      * @return true si la case est valide, false sinon
46      */
47     private boolean estValide(int i, int j) {
48         return i >= 0 && i < m.nbL && j >= 0 && j < m.nbC;
49     }
50
51     /**
52      * Deplace le robot vers les coordonnees (i, j) si la case est valide.
53      *
54      * @param i La coordonnee i de la nouvelle position
55      * @param j La coordonnee j de la nouvelle position
56      */
57     public void vaEn(int i, int j) {
58         if(estValide(i, j)){
59             posx = i;
60             posy = j;
61         }
62     }
63 }
```

```

57     }
58 }
59
60 /**
61  * Methode abstraite qui permet aux robots de parcourir le monde.
62  * Chaque sous-classe de Robot devra implementer cette methode en
63  * fonction de son comportement specifique.
64  */
65 public abstract void parcourir();
66 }

```

## Codes de la classe RobotPollueur

```

1 package Robotpollueurpackage;
2
3 import Mondepackage.Monde;
4 import Robotpackage.Robot;
5
6 /**
7  * Cette classe represente un objet Robotpollueur abstrait, qui est une
8  * sous-classe de Robot.
9  * Un Robotpollueur est capable de polluer les cases du monde.
10 */
11 public abstract class Robotpollueur extends Robot {
12
13     /**
14      * Constructeur qui cree un robot pollueur avec une position donnee
15      * dans le monde donne.
16      *
17      * @param posX La coordonnee x de la position du robot
18      * @param posY La coordonnee y de la position du robot
19      * @param m Le monde dans lequel le robot evolue
20      */
21     public Robotpollueur(int posX, int posY, Monde m) {
22         super(posX, posY, m);
23     }
24
25     /**
26      * Constructeur qui cree un robot pollueur avec une position aleatoire
27      * dans le monde donne.
28      *
29      * @param m Le monde dans lequel le robot evolue
30      */
31     public Robotpollueur(Monde m) {
32         super(m);
33     }
34
35     /**
36      * Methode qui permet au robot pollueur de polluer la case sur
37      * laquelle il se trouve.
38      * Il met un papier gras dans la case.
39      */
40     public void polluer() {
41         m.metPapierGras(posX, posY);
42     }
43
44     /**
45      * Methode abstraite qui permet aux robots pollueurs de parcourir le
46      * monde.
47      * Chaque sous-classe de Robotpollueur devra implementer cette methode
48      * en fonction de son comportement specifique.
49      */

```

```

44     public abstract void parcourir();
45 }

```

## Code de la classe PollueurSauteur.java

```

1  package PollueurSauteurpackage;
2
3  import Mondepacage.Monde;
4  import Robotpollueurpackage.Robotpollueur;
5
6  /**
7   * Cette classe represente un objet PollueurSauteur qui est une
8   * sous-classe de Robotpollueur.
9   * Un PollueurSauteur se deplace dans un monde en sautant d'une colonne a
10   * l'autre et pollue les cases rencontrees.
11   */
12  public class PollueurSauteur extends Robotpollueur {
13      public int colDepart; // Le numero de la colonne de depart du
14      // PollueurSauteur
15      public int deltax; // L'ecart de colonnes entre les sauts
16
17      /**
18       * Constructeur qui cree un PollueurSauteur avec une position initiale
19       * (0, colDepart) dans le monde donne.
20       *
21       * @param colDepart Le numero de la colonne de depart
22       * @param deltax L'ecart de colonnes entre les sauts
23       * @param m Le monde dans lequel le PollueurSauteur evolue
24       */
25      public PollueurSauteur(int colDepart, int deltax, Monde m) {
26          super(0, colDepart, m);
27          this.deltax = deltax;
28      }
29
30      /**
31       * Parcourt le monde en sautant d'une colonne a l'autre et pollue les
32       * cases rencontrees.
33       * Le PollueurSauteur alterne entre les colonnes de depart et les
34       * colonnes decalees de deltax.
35       */
36      public void parcourir() {
37          for(int i = 0; i < m.nbl; i++) {
38              if((i % 2) == 0) {
39                  posx = i;
40                  posy = colDepart;
41                  polluer();
42              } else {
43                  posx = i;
44                  posy = colDepart + deltax;
45                  polluer();
46              }
47          }
48      }
49  }

```

## Codes de la classe PollueurToutDroit.java

```

1  package PollueurToutDroitpackage;
2
3  import Robotpollueurpackage.Robotpollueur;

```

```

4 import Mondepackage.Monde;
5
6 /**
7  * Cette classe represente un objet PollueurToutDroit qui est une
8  * sous-classe de Robotpollueur.
9  * Un PollueurToutDroit se deplace tout droit dans une colonne specifiee
10  * du monde et pollue les cases rencontrees.
11  */
12 public class PollueurToutDroit extends Robotpollueur {
13     public int colDepart; // Le numero de la colonne de depart du
14     PollueurToutDroit
15
16     /**
17      * Constructeur qui cree un PollueurToutDroit avec une position
18      * initiale (0, colDepart) dans le monde donne.
19      *
20      * @param colDepart Le numero de la colonne de depart
21      * @param m Le monde dans lequel le PollueurToutDroit evolue
22      */
23     public PollueurToutDroit(int colDepart, Monde m) {
24         super(0, colDepart, m);
25     }
26
27     /**
28      * Parcourt la colonne specifiee du monde en se deplacant tout droit
29      * et pollue les cases rencontrees.
30      */
31     public void parcourir() {
32         for(int i = 0; i < m.nbl; i++) {
33             posX = i;
34             polluer();
35         }
36     }
37 }

```

## Codes de la classe RobotNettoyeur.java

```

1 package RobotNettoyeurpackage;
2
3 import Robotpackage.Robot;
4 import Mondepackage.Monde;
5
6 /**
7  * Cette classe represente un objet RobotNettoyeur qui est une sous-classe
8  * de Robot.
9  * Un RobotNettoyeur se deplace dans le monde et nettoie les cases sales.
10  */
11 public class RobotNettoyeur extends Robot {
12
13     /**
14      * Constructeur qui cree un RobotNettoyeur avec une position initiale
15      * (0, 0) dans le monde donne.
16      *
17      * @param m Le monde dans lequel le RobotNettoyeur evolue
18      */
19     public RobotNettoyeur(Monde m) {
20         super(0, 0, m);
21     }
22
23     /**
24      * Nettoie la case actuelle si elle est sale.
25      */
26 }

```



```

24     public void nettoyer() {
25         if(m.estSale(posx, posy)){
26             m.prendPapierGras(posx, posy);
27         }
28     }
29
30     /**
31      * Parcourt le monde en se deplacant de maniere specifique et nettoie
32      * les cases sales.
33      */
34     public void parcourir() {
35         for(int i = 0; i < m.nbL; i++) {
36             posx = i;
37             if((i % 2) == 0) {
38                 for(int j = 0; j < m.nbC; j++) {
39                     posy = j;
40                     nettoyer();
41                 }
42             } else {
43                 for(int j = m.nbC - 1; j >= 0; j--) {
44                     posy = j;
45                     nettoyer();
46                 }
47             }
48         }
49     }

```

## Codes de la classe NettoyeurDistrain.java

```

1  package NettoyeurDistrainpackage;
2
3  import Robotpackage.Robot;
4  import Mondepackage.Monde;
5
6  /**
7   * Cette classe represente un objet NettoyeurDistrain qui est une
8   * sous-classe de Robot.
9   */
10
11  public class NettoyeurDistrain extends Robot {
12
13      /**
14       * Constructeur qui cree un NettoyeurDistrain avec une position
15       * initiale (0, 0) dans le monde donne.
16       */
17      @param m Le monde dans lequel le NettoyeurDistrain evolue
18
19      public NettoyeurDistrain(Monde m) {
20          super(0, 0, m);
21      }
22
23      /**
24       * Nettoie la case actuelle du NettoyeurDistrain en enlevant le papier
25       * gras.
26       */
27      public void nettoyer() {
28          m.prendPapierGras(posx, posy);
29      }
30
31      /**
32       * Parcourt le monde en effectuant un mouvement en boustrophedon et
33       * nettoie les cases sales rencontrees.

```

```

29      * Le NettoyeurDistrain alterne entre nettoyer une case sale et ne
      * rien faire pour chaque case rencontree.
30      */
31      public void parcourir() {
32          boolean doitNettoyer = true;
33          int compteur = 0;
34          for(int i = 0; i < m.nbL; i++) {
35              posx = i;
36              if((i % 2) == 0) {
37                  for(int j = 0; j < m.nbC; j++) {
38                      posy = j;
39                      if(m.estSale(posx, posy)) {
40                          if(doitNettoyer) {
41                              nettoyer();
42                              doitNettoyer = false;
43                              compteur++;
44                          } else {
45                              doitNettoyer = true;
46                          }
47                      }
48                  }
49              } else {
50                  for(int j = m.nbC - 1; j >= 0; j--) {
51                      posy = j;
52                      if(m.estSale(posx, posy)) {
53                          if(doitNettoyer) {
54                              nettoyer();
55                              doitNettoyer = false;
56                              compteur++;
57                          } else {
58                              doitNettoyer = true;
59                          }
60                      }
61                  }
62              }
63          }
64          System.out.println("Nombre de case nettoye : " + compteur);
65      }
66  }

```

## Codes de la classe TestRobots.java

```

1  import Mondepackage.Monde;
2  import NettoyeurDistrainpackage.NettoyeurDistrain;
3  import PollueurToutDroitpackage.PollueurToutDroit;
4  import RobotNettoyeurpackage.RobotNettoyeur;
5  import PollueurSauteurpackage.PollueurSauteur;
6
7  /**
8   * Classe de test pour les differentes classes de robots.
9   */
10 public class TestRobots {
11     public static void main(String[] args){
12         // Creation du monde
13         Monde monde = new Monde();
14         System.out.println("\n");
15         System.out.println("+-----+");
16         System.out.println("| CLASSE : Monde |");
17         System.out.println("+-----+\n");
18
19         // Test de la classe Monde
20         System.out.println("Testons la methode metPapierGras aux

```

```

21         coordonnees (1,1) et (1,2)");
22 monde.metPapierGras(1, 1);
23 monde.metPapierGras(1, 2);
24 System.out.println(monde.toString());
25 System.out.print("Testons la methode estSale aux coordonnees (1,1)
26 : ");
27 System.out.println(monde.estSale(1, 1));
28 System.out.println("Le nombre de papiers gras dans le monde : " +
29 monde.nbPapiersGras());
30 System.out.println("Testons la methode prendPapierGras aux
31 coordonnees (1,1)");
32 monde.prendPapierGras(1, 1);
33 System.out.println(monde.toString());
34 System.out.println("Le nombre de papiers gras dans le monde : " +
35 monde.nbPapiersGras());
36 System.out.println("\n\n\n");
37
38 //Test de la classe PollueurToutDroit
39 System.out.println("+-----+");
40 System.out.println("| CLASSE : PollueurToutDroit |");
41 System.out.println("+-----+\n");
42
43 // Creation et test d'un PollueurToutDroit
44 PollueurToutDroit p = new PollueurToutDroit(3, monde);
45 System.out.println("Testons la methode parcourir avec colDepart =
46 3");
47 p.parcourir();
48 System.out.println(monde.toString());
49 System.out.println("\n\n\n");
50
51 // Test de la classe PollueurSauteur
52 System.out.println("+-----+");
53 System.out.println("| CLASSE : PollueurSauteur |");
54 System.out.println("+-----+\n");
55
56 // Creation et test d'un PollueurSauteur
57 PollueurSauteur polSauteur = new PollueurSauteur(0, 1, monde);
58 System.out.println("Testons la methode parcourir avec colDepart =
59 0 et deltax = 1");
60 polSauteur.parcourir();
61 System.out.println(monde.toString());
62 System.out.println("\n\n\n");
63
64 // Test de la classe RobotNettoyeur
65 System.out.println("+-----+");
66 System.out.println("| CLASSE : RobotNettoyeur |");
67 System.out.println("+-----+\n");
68
69 // Creation et test d'un RobotNettoyeur
70 System.out.println("Testons la methode parcourir :");
71 System.out.println("Affichons d'abord la matrice ");
72 System.out.println(monde.toString());
73 RobotNettoyeur robNettoyeur = new RobotNettoyeur(monde);
74 System.out.println("La matrice apres nettoyage ");
75 robNettoyeur.parcourir();
76 System.out.println(monde.toString());
77 System.out.println("\n\n\n");
78
79 //Test de la classe NettoyeurDistrait

```

```

77         System.out.println("+-----+");
78         System.out.println("| CLASSE : NettoyeurDistrait |");
79         System.out.println("+-----+\n");
80
81         //Creation et test d'un NettoyeurDistrait
82         System.out.println("Testons la methode parcourir : ");
83         System.out.println("Affichons d'abord la matrice ");
84         p.parcourir();
85         System.out.println(monde.toString());
86         NettoyeurDistrait netDistrait = new NettoyeurDistrait(monde);
87         System.out.println("La matrice apres nettoyage ");
88         netDistrait.parcourir();
89         System.out.println(monde.toString());
90         System.out.println("\n\n\n");
91     }
92 }

```

## Tests et Captures

### Exercice 1 :

Il s'agissait ici de créer la classe Monde avec les différentes méthodes citées ci-dessus :

```

1 // Creation du monde
2 Monde monde = new Monde();
3 System.out.println("\n");
4 System.out.println("+-----+");
5 System.out.println("| CLASSE : Monde |");
6 System.out.println("+-----+\n");
7
8 // Test de la classe Monde
9 System.out.println("Testons la methode metPapierGras aux coordonnees (1,1)
   et (1,2)");
10 monde.metPapierGras(1, 1);
11 monde.metPapierGras(1, 2);
12 System.out.println(monde.toString());
13 System.out.print("Testons la methode estSale aux coordonnees (1,1) : ");
14 System.out.println(monde.estSale(1, 1));
15 System.out.println("Le nombre de papiers gras dans le monde : " +
   monde.nbPapiersGras());
16 }

```

Dans ce code nous avons créé un objet monde en utilisant le constructeur sans paramètre. Par la suite, nous appelons la méthode **metPapierGras** sur l'objet créé pour qu'il puisse mettre des papiers gras sur les positions données en paramètres. On affiche le monde en appelant la méthode **toString**. Après nous appelons la méthode **estSale** pour vérifier si la case (1,1) est sale. Enfin, nous appelons la méthode **nbPapiersGras** pour compter le nombre de papiers gras dans le monde.

```

D:\DIC1\Semestre2\P00-JAVA\I.Fall\LesRobotsPollueurs>javac TestRobots.java
D:\DIC1\Semestre2\P00-JAVA\I.Fall\LesRobotsPollueurs>java TestRobots

+-----+
| CLASSE : Monde |
+-----+

Testons la methode metPapierGras aux coordonnees (1,1) et (1,2)
. . . . .
. o o . . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .

Testons la methode estSale aux coordonnees (1,1) : true
Le nombre de papiers gras dans le monde : 2

```

On voit bien l'appel de la méthode **metPapierGras** (2 fois) a permis de mettre des papiers gras dans le monde dans les différentes positions données en paramètre.

En bas on voit **true**, c'est parceque l'appel de la méthode **estSale** avec comme paramètre (1,1) montre qu'au niveau de cette case il y a présence de papier gras.

On voit bien que le nombre de papiers gras dans le monde est 2.

```

1 System.out.println("Testons la methode prendPapierGras aux coordonnees
  (1,1)");
2 monde.prendPapierGras(1, 1);
3 System.out.println(monde.toString());
4 System.out.println("Le nombre de papiers gras dans le monde : " +
  monde.nbPapiersGras());
5 System.out.println("\n\n\n");
6 }

```

Maintenant dans le code ci-dessus, on va essayer de tester la méthode **prendPapierGras**. Puis on affiche le monde et compte le nombre de papiers gras.

```

Testons la methode prendPapierGras aux coordonnees (1,1)
. . . . .
. . o . . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .

Le nombre de papiers gras dans le monde : 1

```

On voit bien que la méthode **prendPapierGras** enlève le papier gras de la position (1,1) et le nombre de papiers gras revient 1.

## Exercice 2 :

Il s'agissait ici de créer la classe **Robot** (classe abstraite) avec les différentes méthodes citées ci-dessus.

Puisque la classe **Robot** est abstraite, on ne peut pas l'instancier.

## Exercice 3 :

---

Il s'agissait ici de créer la classe **RobotPollueur** (classe abstraite) avec les différentes méthodes citées ci-dessus.

Cette classe également est abstraite, on ne peut pas l'instancier.

### Exercice 4 :

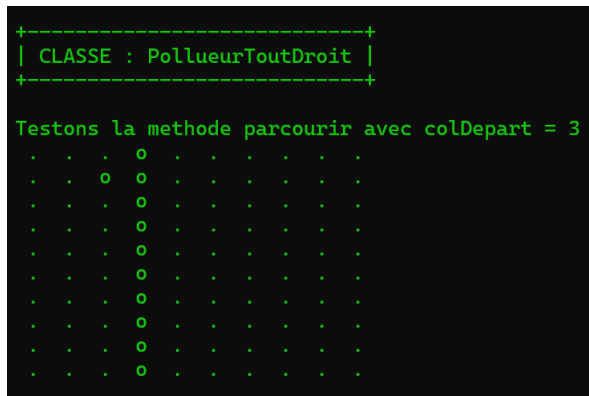
Il s'agissait ici de créer la classe **PollueurToutDroit** avec les différentes méthodes citées ci-dessus.

```
1 //Test de la classe PollueurToutDroit
2 System.out.println("+-----+");
3 System.out.println("| CLASSE : PollueurToutDroit |");
4 System.out.println("+-----+\n");
5
6 // Creation et test d'un PollueurToutDroit
7 PollueurToutDroit p = new PollueurToutDroit(3, monde);
8 System.out.println("Testons la methode parcourir avec colDepart = 3");
9 p.parcourir();
10 System.out.println(monde.toString());
11 System.out.println("\n\n\n");
```

Dans ce code nous avons créé un objet **p**.

Par la suite, nous appelons la méthode **parcourir** sur l'objet créé pour qu'il puisse parcourir la colonne spécifiée (3) du monde en se déplaçant tout droit et pollue les cases rencontrées..

Enfin, on affiche le monde grâce à la méthode **toString**



On voit bien que toute les cases de la colonnes sont polluées.

### Exercice 5 :

Il s'agissait ici de créer la classe **PollueurSauteur** avec les différentes méthodes citées ci-dessus.

```
1 // Test de la classe PollueurSauteur
2 System.out.println("+-----+");
3 System.out.println("| CLASSE : PollueurSauteur |");
4 System.out.println("+-----+\n");
5
6 // Creation et test d'un PollueurSauteur
7 PollueurSauteur polSauteur = new PollueurSauteur(0, 1, monde);
8 System.out.println("Testons la methode parcourir avec colDepart = 0 et
9   deltax = 1");
9 polSauteur.parcourir();
10 System.out.println(monde.toString());
11 System.out.println("\n\n\n");
```

Dans ce code nous avons créé un objet **polSauteur**.

Par la suite, nous appelons la méthode **parcourir** sur l'objet créé pour que le robot se deplace dans le monde en sautant d'une colonne a l'autre et pollue les cases rencontrées

---

Enfin, on affiche le monde grâce à la méthode **toString**

```
+-----+
| CLASSE : PollueurSauteur |
+-----+

Testons la methode parcourir avec colDepart = 0 et deltax = 1
O . . O . . . . .
. O O O . . . . .
O . . O . . . . .
. O . O . . . . .
O . . O . . . . .
. O . O . . . . .
O . . O . . . . .
. O . O . . . . .
O . . O . . . . .
. O . O . . . . .
```

On voit bien que le robot se deplace dans le monde en sautant d'une colonne a l'autre et pollue les cases rencontrées

## Exercice 6 :

Il s'agissait ici de créer la classe **RobotNettoyeur** avec les différentes méthodes citées ci-dessus.

```
1 // Test de la classe RobotNettoyeur
2 System.out.println("-----+");
3 System.out.println("| CLASSE : RobotNettoyeur |");
4 System.out.println("-----+\n");
5
6 // Creation et test d'un RobotNettoyeur
7 System.out.println("Testons la methode parcourir :");
8 System.out.println("Affichons d'abord la matrice ");
9 System.out.println(monde.toString());
10 RobotNettoyeur robNettoyeur = new RobotNettoyeur(monde);
11 System.out.println("La matrice apres nettoyage ");
12 robNettoyeur.parcourir();
13 System.out.println(monde.toString());
14 System.out.println("\n\n\n");
```

Dans ce code nous avons créé un objet **robNettoyeur**.

Par la suite, nous redéfinissons la méthode **parcourir** sur l'objet créé pour qu'il se déplace dans le monde et nettoie les cases sales.

Enfin, on affiche le monde grâce à la méthode **toString**

```
+-----+
| CLASSE : RobotNettoyeur |
+-----+

Testons la methode parcourir :
Affichons d'abord la matrice

O . . O . . . . .
. O O O . . . . .
O . . O . . . . .
. O . O . . . . .
O . . O . . . . .
. O . O . . . . .
O . . O . . . . .
. O . O . . . . .
O . . O . . . . .
. O . O . . . . .

La matrice apres nettoyage

. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

On voit bien comment est le monde avant et après l'appel de la méthode **parcourir**.

### Exercice 6 :

Il s'agissait ici de créer la classe **NettoyeurDistrain** avec les différentes méthodes citées ci-dessus.

```
1 //Test de la classe NettoyeurDistrain
2 System.out.println("+-----+");
3 System.out.println("| CLASSE : NettoyeurDistrain |");
4 System.out.println("+-----+\n");
5
6 //Creation et test d'un NettoyeurDistrain
7 System.out.println("Testons la methode parcourir : ");
8 System.out.println("Affichons d'abord la matrice ");
9 p.parcourir();
10 System.out.println(monde.toString());
11 NettoyeurDistrain netDistrain = new NettoyeurDistrain(monde);
12 System.out.println("La matrice apres nettoyage ");
13 netDistrain.parcourir();
14 System.out.println(monde.toString());
15 System.out.println("\n\n\n");
```

Dans ce code nous avons créé un objet **netDistrain**.  
Par la suite, nous redéfinissons la méthode **parcourir** sur l'objet créé.  
Cette méthode alterne entre nettoyer une case sale et ne rien faire pour chaque case rencontrée.  
Enfin, on affiche le monde grâce à la méthode **toString**



```
+-----+
| CLASSE : NettoyeurDistrait |
+-----+

Testons la methode parcourir :
Affichons d'abord la matrice

. . . O . . . . .
. . . O . . . . .
. . . O . . . . .
. . . O . . . . .
. . . O . . . . .
. . . O . . . . .
. . . O . . . . .
. . . O . . . . .
. . . O . . . . .

La matrice apres nettoyage
Nombre de case nettoyees : 5

. . . . .
. . . O . . . . .
. . . . .
. . . O . . . . .
. . . . .
. . . O . . . . .
. . . . .
. . . O . . . . .
. . . . .
```

On voit bien que le nettoyeur n'enlève qu'un papier sur deux.

Archives :

Commandes :

jar cvf RobotsPollueursArchives Mondepacage Robotpackage PollueurToutDroitpackage Robotpollueurpackage PollueurSauteurpackage RobotNettoyeurpackage NettoyeurDistraitpackage

```
D:\DIC1\Semestre2\P00-JAVA\I.Fall\LesRobotsPollueurs>jar cvf RobotsPollueursArchives Mondepacage Robotpackage PollueurToutDroitpackage Robotpollueurpackage PollueurSauteurpackage RobotNettoyeurpackage NettoyeurDistraitpackage
added manifest
adding: Mondepacage/(in = 0) (out= 0)(stored 0%)
adding: Mondepacage/Monde.class(in = 1318) (out= 775)(deflated 41%)
adding: Mondepacage/Monde.java(in = 3343) (out= 871)(deflated 73%)
adding: Robotpackage/(in = 0) (out= 0)(stored 0%)
adding: Robotpackage/Robot.class(in = 805) (out= 523)(deflated 35%)
adding: Robotpackage/Robot.java(in = 1970) (out= 675)(deflated 65%)
adding: PollueurToutDroitpackage/(in = 0) (out= 0)(stored 0%)
adding: PollueurToutDroitpackage/PollueurToutDroit.class(in = 577) (out= 368)(deflated 36%)
adding: PollueurToutDroitpackage/PollueurToutDroit.java(in = 1114) (out= 432)(deflated 61%)
adding: Robotpollueurpackage/(in = 0) (out= 0)(stored 0%)
adding: Robotpollueurpackage/Robotpollueur.class(in = 563) (out= 338)(deflated 39%)
adding: Robotpollueurpackage/Robotpollueur.java(in = 1447) (out= 503)(deflated 65%)
adding: PollueurSauteurpackage/(in = 0) (out= 0)(stored 0%)
adding: PollueurSauteurpackage/PollueurSauteur.class(in = 668) (out= 436)(deflated 34%)
adding: PollueurSauteurpackage/PollueurSauteur.java(in = 1565) (out= 535)(deflated 65%)
adding: RobotNettoyeurpackage/(in = 0) (out= 0)(stored 0%)
adding: RobotNettoyeurpackage/RobotNettoyeur.class(in = 827) (out= 527)(deflated 36%)
adding: RobotNettoyeurpackage/RobotNettoyeur.java(in = 1344) (out= 495)(deflated 63%)
adding: RobotNettoyeurpackage/texput.log(in = 773) (out= 472)(deflated 38%)
adding: NettoyeurDistraitpackage/(in = 0) (out= 0)(stored 0%)
adding: NettoyeurDistraitpackage/NettoyeurDistrait.class(in = 1536) (out= 885)(deflated 42%)
adding: NettoyeurDistraitpackage/NettoyeurDistrait.java(in = 2185) (out= 650)(deflated 70%)
```

---

## Documentation API de chaque projet (package) :

### Commandes :

```
javadoc -d docs -sourcepath D:\DIC1\Semestre2\POO-JAVA\I.Fall\LesRobotsPollueurs  
Mondepackage Robotpackage PollueurToutDroitpackage Robotpollueurpackage Pol-  
lueurSauteurpackage RobotNettoyeurpackage NettoyeurDistraitpackage
```