- Pas de document autorisé, téléphones portables éteints
- Penser à inscrire votre numéro d'anonymat de manière lisible sur la copie
- Le barème est donné à titre indicatif. ATTENTION : répartition des points spécifique à cette année (22 pts de cours, 38 pts de problème)

Questions de cours (22 pts)

Exercice 1 – Syntaxe de base JAVA (1 pt)

Q 1.1 (1 pt) Soit l'instruction suivante :

```
1 System.out.println("coucou");
```

A quoi correspondent : `System`, `out` et `println` ? Les choix possibles de réponses sont : au nom d'une classe, au nom d'une variable d'instance, à une variable statique, à un appel à une méthode d'instance, à un appel à une méthode statique. Justifier brièvement.

- System = classe
- out = variable static (public)
- println = méthode d'instance (public)

Exercice 2 – Instances et références (12 pts)

Soit les classes suivantes `A` et `B` et le `main` associé :

```
1 public class A{
2   private int i;
3   public A(int i){ this.i=i; }
4   public A clone(){return new A(i);}
5 }
```

```
1 public class B{
2   private A a;
3   public B(A a){ this.a=a; }
4   public B clone(){return new B(a);}
5 }
```

```
1 public static void main(String[] args){
2   A a1 = new A(2);
3   A a2 = a1.clone();
4   A a3 = a1;
5   if(a1 == a2) System.out.println("a1 == a2");
6   if(a2 == a3) System.out.println("a2 == a3");
7   if(a3 == a1) System.out.println("a3 == a1");
8 }
```

Q 2.1 (1 pt) Combien y a-t-il d'instances de la classe `A` créées lors de l'exécution du `main` ?

Réponse : 2

Q 2.2 (1 pt) Qu'est-ce qui s'affiche lors de l'exécution du programme ?

Réponse : a3 == a1

Nous ajoutons les instructions suivantes dans le `main` précédent.

```
1  if(a1.equals(a2)) System.out.println("a1.equals(a2)");
2  if(a2.equals(a3)) System.out.println("a2.equals(a3)");
3  if(a3.equals(a1)) System.out.println("a3.equals(a1)");
```

Q 2.3 (1 pt) Le programme compile-t-il toujours ? Dans l'affirmative, qu'est-ce qui s'affiche lors de l'exécution ? (Justifier brièvement)

Réponse : le pg compile. Même réponse que pour == (définition par défaut de equals héritée de la classe Object)

Q 2.4 (2,5 pts) Donner le code de la méthode standard `boolean equals(Object o)` à ajouter dans la classe `A` pour tester l'égalité structurelle entre 2 instances.

```
1 public boolean equals(Object o){
2   if(o == this) return true; // facultatif
3   if(o == null) return false;
4   if(o.getClass() != getClass()) return false; // ou usage de instanceof
5
6   A o2 = (A) o;
7   return o2.i == i;
8 }
```

Etudions le code du programme suivant :

```
1 public static void main(String[] args){
2   A a1 = new A(2);
3   B b1 = new B(a1);
4   B b2 = b1.clone();
5 }
```

Q 2.5 (2 pts) Donner la représentation de la mémoire à la fin de l'exécution du code. Combien d'instances de `A` et `B` sont créées ?

Réponse :
- Variable a1 de type A référence une instance de type A
- Variable b1 de type B référence une instance de type B
- Variable b2 de type B référence une instance de type B
- MAIS : b1 et b2 pointent sur a1

2 instances de B, 1 de A

Q 2.6 (1 pts) La question précédente met en évidence le mauvais fonctionnement de la méthode de clonage de la classe `B` : proposer une amélioration.

```
1 // cloner A!
2 public B clone(){return new B(a.clone());}
```

Q 2.7 (1,5 pts) Donner le code de la méthode standard `boolean equals(Object o)` à ajouter dans la classe `B` pour tester l'égalité structurelle entre 2 instances.

```
1 public boolean equals(Object o){
2   if(o == this) return true;
3   if(o == null) return false;
4   if(o.getClass() != getClass()) return false;
5   B o2 = (B) o;
6
7   return o2.a.equals(a);  // penser à equals !
8 }
```

Encore une étude de cas :

```
public static void main(String[] args){
  A[] tab = new A[5];
  for(int i=0; i<=5; i++) System.out.println(tab[i]);
}
```

Q 2.8 (2 pt) Le programme compile-t-il ? Dans l'affirmative, que se passe-t-il lors de l'exécution ?

- OUI, ça compile
- Affichage null null... + IndexArrayOutOfBoundException

Exercice 3 – Redéfinition complexe (3 pts)

```
public class Mere {
  public Mere(){}
  public void maMethode(double d){
    System.out.println("arg : double = "+
        d);
  }
}
public class Fille extends Mere{
  public Fille(){}
  public void maMethode(int i){
    System.out.println("arg : int = "+i);
  }
}
```

```
public static void main(String[] args) {
  Mere m = new Mere();
  Fille f = new Fille();
  Mere mf = new Fille();

  m.maMethode(2);
  f.maMethode(2);
  mf.maMethode(2);

  m.maMethode(2.5);
  f.maMethode(2.5);
  mf.maMethode(2.5);
}
```

Q 3.1 (2 pts) Quels sont les affichages observés dans la console suite à l'exécution de ce programme ? NB : 5 réponses sont assez triviales mais 1 ligne est plus délicate. Expliquer en détail ce qui se passe avec cette ligne.

Réponse :

```
arg : double = 2.0
arg : int = 2
arg : double = 2.0 // le compilateur a sélectionné la méthode avec arg (double)
arg : double = 2.5
arg : double = 2.5
arg : double = 2.5
```

Q 3.2 (1 pt) Nous ajoutons une implémentation de la méthode `public double maMethode(double d)` dans la classe `Mere` : le code compile-t-il toujours ?

Réponse : ça ne compile plus, impossible de faire cohabiter 2 méthodes de même signature (le type de retour n'est pas pris en compte pour différentier deux méthodes)

Exercice 4 – Exceptions (6pts)

Le code suivant ne s'occupe pas des exceptions :

```
1 public class Pile {
2   private static final int TAILLE = 10;
3   private int[] contenu;
4   private int niveau;
5
6   public Pile(){
7     contenu = new int[TAILLE];
8     niveau  = 0;
9   }
10  public void push(int i){
11    contenu[niveau] = i;
12    niveau++;
13  }
14  public int pop(){
15    int r = contenu[niveau];
16    niveau --;
17    return r;
18  }
19 }
```

```
1 class PileException extends Exception{
2     public PileException() {
3         super();
4     }
5     public PileException(String message)
          {
6         super(message);
7     }
8 }
```

Q 4.1 (3pts) Proposer des nouvelles versions pour les méthodes `push` et `pop` qui lèvent une `PileException` lorsque nous tentons :
– d'appeler `push` alors que la pile contient déjà 10 éléments
– d'appeler `pop` alors que la pile est vide (0 éléments dans `contenu`)
NB : les exceptions sont déléguées, elles ne sont pas traitées au niveau local.

```
1 public void push(int i) throws PileException{
2     if(niveau == TAILLE)
3       throw new PileException("pile␣pleine,␣impossible␣d'ajouter␣un␣élément");
4     contenu[niveau] = i;
5     niveau++;
6   }
7   public int pop() throws PileException{
8    if(niveau == 0)
9       throw new PileException("pile␣vide,␣impossible␣d'enlever␣un␣élément");
10    int r = contenu[niveau];
11    niveau --;
12    return r;
13  }
```

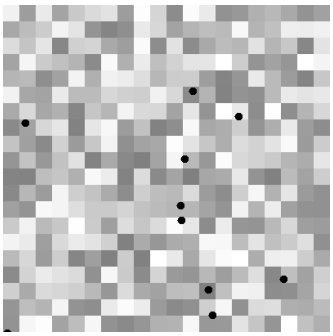
Q 4.2 (3pts) Proposer le code d'une classe `Test`, contenant un `main` qui :

1. crée une pile,
2. la remplit jusqu'à ce qu'elle déborde,
3. attrape l'exception et affiche le message : `"pile pleine, impossible d'ajouter un élément"`,
4. enlève un élément de la pile et l'affiche.

```
1 public class Test{
2  public static void main(String[] args){
3    Pile p = new Pile();
4    try{
5      for(int i=0; i<12; i++) p.push(i);
6     } catch(PileException e){
7       System.out.println( e.getMessage());
8     }
9     try{
10     int r = p.pop();
11     System.out.println( r);
12    } catch(PileException e){
13      System.out.println( e.getMessage());
14    }
15
16 }
17 }
```

Problème : fourmis et modèles de votes (38 pts)

Le but de cet exercice est de modéliser le comportement des électeurs lors d'un scrutin binaire (vote *pour* ou *contre* quelque chose, deux possibilités). Les votants sont des fourmis qui ont une opinion et qui votent toutes. Les fourmis se déplacent sur le terrain. Certaines fourmis laissent des traces (phéromones) sur le terrain correspondant à leur opinion, toutes les fourmis lisent ces traces et peuvent être influencées. L'outil de simulation permet de suivre l'évolution des opinions en réalisant des sondages sur l'ensemble de la population de fourmis.
Nous faisons les hypothèses suivantes :
– l'opinion d'une fourmi est un réel compris entre 0 et 1 (exclu),
– le vote d'une fourmi est un booléen : (`true` si opinion ≥ 0.5, `false` sinon). Nous considérons que `true` signifie que la fourmi vote *pour* et `false` qu'elle vote *contre*, le booléen permet d'exprimer les deux possibilités de vote.



– Le terrain est constitué d'une matrice dont chaque case synthétise l'opinion des fourmis qui sont passées par là. Dans la représentation de gauche, la couleur des cases indiquent si elles contiennent des traces plutôt associées à l'opinion *pour* (blanc) ou *contre* (noir).
– Le terrain a pour dimension `DIMxDIM`.
– Les fourmis ont des coordonnées réelles comprises dans $[0,\ DIM[$, elles peuvent se déplacer à l'intérieur d'une case de terrain : par exemple, la fourmi `f` de coordonnées $(2.4, 0.6)$ et d'opinion 0.3 se trouve dans la case $(2, 0)$ et vote *contre* (=`false` car opinion < 0.5).
– A n'importe quel moment, je peux interroger mes fourmis pour savoir le pourcentage de vote *pour* par rapport au vote *contre*.

FIGURE 1 – Image associée au terrain et fourmis se déplaçant sur le terrain

Pour l'instant, il suffit de savoir que toutes les fourmis hériteront de la classe `Fourmi` et possèderont les méthodes suivantes :
- `void bouger()` : déplacement de la fourmi
- `void agir(Monde m)` : interaction de la fourmi avec le monde (exemples : lecture des traces sur le terrain, dépôt de traces...)
- `boolean voter()` : retourne `true` si la fourmi vote *pour*, `false` sinon (toutes les fourmis votent)

Les différents types de fourmis seront détaillés plus tard.

Exercice 5 – Outils (2pts)

Par la suite, nous aurons besoin d'une méthode permettant de passer d'un tableau de réels à un tableau d'entier. Nous souhaitons développer la méthode `int[] convToInt(double[] tab)` dans la classe `Outils`. Pour plus de commodité, nous souhaitons invoquer cette méthode sans instancier d'objet de la classe `Outils`, en utilisant la syntaxe suivante :

```
1 // avec coordAsDouble qui est de type double[]
2 int[] coordAsInt = Outils.convToInt(coordAsDouble);
```

Cette méthode fonctionnera quelle que soit la dimension du tableau fourni en entrée.
Donner le code de la classe `Outils` (qui ne contient qu'une méthode).

```
1     public static int[] convToInt(double[] tab){
2         int[] retour = new int[tab.length];
3         for(int i=0; i<tab.length; i++) retour[i] = (int) tab[i];
4         return retour;
5     }
```

Exercice 6 – Monde (14 pts)

Le monde contient les attributs suivants :
- `DIM` une constante publique donnant la dimension du monde (carré). Cette constante est égale à 20.
- `EPSMONDE` une constante privée égale à 0.01 (pour la mise à jour des cases lorsqu'une fourmi laisse des traces).
- `population` un tableau de `Fourmi` à donner au constructeur pour la création du monde.
- `terrain` une matrice de réels de taille `DIMxDIM` où chaque case est initialisée aléatoirement (entre 0 et 1 exclu)
- `sondage` un tableau de 2 réels donnant le pourcentage actuel de vote *pour* et de vote *contre*.

Q 6.1 (2 pts) Classe `Monde`. Donner la signature de la classe et déclarer les attributs (vous ajouterez le code des questions suivantes à la suite).

```
1 public class Monde {
2     public static final int DIM = 20;
3     public static final double EPSMONDE = 0.01;
4
5     public Fourmi[] population;
6     public double[][] terrain;
7     public double[] sondage;
8
```

```
9     private void miseAJourSondage(){
10        int nbPos = 0;
11        for(Fourmi f:population)
12            if(f.vote()) nbPos ++;
13
14        sondage[0] = nbPos/(double) population.length; // attention à éviter la
              division entière
15        sondage[1] = 1-nbPos/(double) population.length;
16    }
17
18    public Monde(Fourmi[] population) {
19        sondage = new double[2];
20        this.population = population;
21        terrain = new double[DIM][DIM];
22        for(int i=0; i<terrain.length; i++)
23            for(int j=0; j<terrain[i].length; j++)
24                terrain[i][j] = Math.random();
25
26        miseAJourSondage(); // ne pas oublier
27    }
28
29    public void setTrace(double[] pos, boolean vote){
30        int i = (int) pos[0]; int j = (int) pos[1];
31        terrain[i][j] += (vote?1:-1) * EPSMONDE;
32        terrain[i][j] = Math.min(1, terrain[i][j]); // check bounds
33        terrain[i][j] = Math.max(0, terrain[i][j]);
34    }
35
36    public boolean getTrace(double[] pos){
37        return terrain[(int) pos[0]][(int) pos[1]]>0.5;
38    }
39
40    public double[] getSondage(){
41        return sondage;
42    }
43
44
45    public void update(){
46        // faire bouger les fourmis + action
47        for(Fourmi f:population){
48            f.bouger(); // appel aux méthodes abstraites
49            f.agir(this);
50        }
51        miseAJourSondage(); //  penser à mettre à jour le sondage
52    }
53
54    public String toString(){ // NON DEMANDE DANS L'ENONCE
55        StringBuffer sb = new StringBuffer();
56        for(int i=0; i<terrain.length; i++){
57            for(int j=0; j<terrain[i].length; j++)
58                sb.append(String.format("%3.2f␣",terrain[i][j]));
59            sb.append("\n");
60        }
61        return sb.toString();
62    }
63
```

```
64      public BufferedImage toImage(int fact){ // NON DEMANDE DANS L'ENONCE
65          int ref = 128;
66          BufferedImage im = new BufferedImage(DIM*fact, DIM*fact, BufferedImage.
                TYPE_INT_ARGB);
67          for(int i=0; i<terrain.length; i++)
68              for(int j=0; j<terrain[i].length; j++)
69                  for(int k=0; k<fact; k++)
70                      for(int m=0; m<fact; m++)
71                          im.setRGB(i*fact+k, j*fact+m, (new Color(255 - (int) (
                                terrain[i][j]*ref), 255 - (int) (terrain[i][j]*ref),
72                                  255 - (int) (terrain[i][j]*ref)).getRGB()));
73
74          for(Fourmi f:population)
75              f.draw(im,fact);
76
77          return im;
78      }
79  }
```

Q 6.2 (3 pts) Méthode `void miseAJourSondage()`. Donner le code de cette méthode qui permet de mettre à jour l'attribut `sondage` en fonction du vote des fourmis de la `population`. Cette méthode comptabilise le nombre de vote *pour* et *contre* dans la population (respectivement np et nc) et met à jour `sondage` de sorte que `sondage[0]` $= \frac{np}{np+nc}$ et `sondage[1]` $= \frac{nc}{np+nc}$.
Aide : en fonction du type des variables que vous choisissez, il y a un risque que les divisions donnent 0, à vous de proposer une bonne solution.

Q 6.3 (2,5 pts) Constructeur. Donner le code du constructeur qui prend en argument une population de fourmis, initialise `terrain` aléatoirement (toutes les cases entre 0 et 1 exclu) et initialise `sondage` en fonction de l'opinion initiale des fourmis.

Q 6.4 (0,5 pt) Accesseur. Donner le code de `double[] getSondage()` qui retourne l'attribut `sondage`.

Q 6.5 (3 pts) Interactions avec le monde.
- Donner le code de `boolean getTrace(double[] coordonnees)` qui retourne la trace laissée par les fourmis sur la case repérée par `coordonnees`. Dans la pratique, cette méthode permettra à une fourmi (dont la position sera fournie en argument) de récupérer les traces laissées par ses prédécesseurs sur sa position actuelle.
  NB1 : la trace lue est binaire (`true` si le `terrain` associé aux coordonnées est ≥ 0.5, `false` sinon)
  NB2 : pour accéder à une case de `terrain`, les coordonnées doivent être entières, à vous de faire les conversions (vous pouvez utiliser la méthode de l'exercice 5).
- Donner le code de `void setTrace(double[] position, boolean vote)` qui permet à une fourmi de déposer une trace à la position où elle se trouve.
  Formule de mise à jour du `terrain` à la position de la fourmi :

$$trace = trace + \begin{cases} EPSMONDE & \text{si } \texttt{vote} \text{ est } \texttt{true} \\ -EPSMONDE & \text{si } \texttt{vote} \text{ est } \texttt{false} \end{cases}$$

  Cette formule permet de modifier la trace sur le terrain pour la rendre plus proche de l'opinion de la fourmi.

  Attention : il faut vérifier que l'opinion est toujours comprise entre 0 et 1 après mise à jour (et ramener la valeur entre 0 et 1 inclus si ce n'est pas le cas).

Q 6.6 (3 pts) Evolution du monde. Le monde se met à jour itérativement à travers la méthode `void update()`. Dans cette méthode, toutes les fourmis sont passées en revue, elles bougent (appel à `bouger()`) puis elles agissent (appel à `agir(Monde m)`). Après quoi, il est nécessaire de mettre à jour le sondage.

Exercice 7 – Fourmis (18 pts)

Afin de modéliser les particularités des votants, nous distinguerons 3 types de fourmis concrètes :

1. les **flottantes** (qui sont indécises, lisent les traces sur le terrain et modifient leur opinion mais qui ne laissent pas de trace sur le terrain)
2. les **évangélisatrices** (qui sont convaincues de leur opinion, laissent des traces sur le terrain mais ne modifient jamais leur opinion)
3. les **contradicteurs** (qui sont de l'avis opposé à la majorité -quelle qu'elle soit-)

Les **flottantes** sont des fourmis standards, les **évangélisatrices** et les **contradicteurs** sont des fourmis **actives** qui sont capables de laisser des traces sur le terrain.

Q 7.1 (1 pt) Donner une arborescence classes pour l'organisation des fourmis. Préciser les classes qui sont abstraites ou pas. Nous privilégierons une architecture à 5 classes de fourmis (fortement conseillé, mais pas obligatoire).

Fourmi (ABS)
FourmiActive (ABS)
FourmiFlottante
FourmiEvang
FourmiContr

Q 7.2 (6 pts) Donner le code de la classe `Fourmi` respectant les directives suivantes.
Une `Fourmi` a les caractéristiques suivantes :

- Attributs : `position` (tableau de 2 réels compris entre 0 et DIM -exclu-) et `opinion` (réel entre 0 et 1 -exclu-)
- Constante privée : `EPSFOURMI = 0.2` pour la mise à jour de l'opinion de la fourmi
- Constructeur prenant en argument une position et une opinion
- Accesseur sur la position `getPosition`, méthode `voter` retournant un booléen (`true` si opinion ≥ 0.5, `false` sinon),
- Méthode `bouger` sans argument, permettant un mouvement aléatoire de la Fourmi : les deux cases du tableau `position` sont modifiées en ajoutant deux réels tirés aléatoirement entre -1 et 1 (ne vous occupez pas de savoir si 1 et -1 sont inclus ou non). Afin de toujours rester dans le monde, nous utiliserons l'hypothèse du monde torique :

```
1 // Etant donnée DIM, la dimension du monde à récupérer par vos soins
2 position[0] = (position[0] + DIM) % DIM; // monde torique
3 position[1] = (position[1] + DIM) % DIM;
```

- Méthode `void agir(Monde m)`, permettant à chaque fourmi de faire son travail.
  NB : chaque type de Fourmi aura une action spécifique dans les classes filles.
- Pour les classes descendantes de `Fourmi` (et pour elles seulement), deux méthodes supplémentaires sont nécessaires pour modifier l'opinion d'une fourmi :
  - `void setOpinion(double opinion)` : pour affecter une nouvelle opinion à une fourmi.
  - `void modifierOpinion(boolean influence)` : une méthode de modification de l'opinion. Si la fourmi est influencée vers `true`, alors son opinion est incrémentée de `EPSFOURMI`, sinon, elle est décrémentée de `EPSFOURMI`.
    Attention : l'opinion de la fourmi doit toujours rester comprise entre 0 et 1.

```java
public abstract class Fourmi {
    public static final double EPSFOURMI = 0.2;

    private double[] position;
    private double opinion;

    public Fourmi(double[] position, double opinion) {
        this.position = position;
        this.opinion = opinion;
    }

    public  void bouger(){ // 1 pt
        position[0] += (Math.random()-0.5) * 2;
        position[1] += (Math.random()-0.5) * 2;

        position[0] = (position[0] + Monde.DIM) % Monde.DIM; // monde torique
        position[1] = (position[1] + Monde.DIM) % Monde.DIM;
    }

    public abstract void agir(Monde m); // ABS


    protected void modifierOpinion(boolean mod){
        opinion += (mod?1:-1)*EPSFOURMI;
        opinion = Math.min(1, opinion); // check bounds
        opinion = Math.max(0, opinion);
    }

    protected void setOpinion(double newOpi){
        opinion = newOpi;
    }

    public boolean voter(){
        return opinion > 0.5;
    }

    public double[] getPosition(){
        return position;
    }

    public String toString(){ // NON DEMANDE
        return "F:␣o="+String.format("%3.2f",opinion);
    }

    public void draw(BufferedImage im, int fact) { // NON DEMANDE
        //        Graphics2D g = (Graphics2D) im.getGraphics();
        Graphics g = im.getGraphics();
        g.setColor(Color.black);
        //        g.setStroke(new BasicStroke(3));
        g.fillOval((int) (getPosition()[0]*fact), (int) (getPosition()[1]*fact),
            fact/2,fact/2);

    }

}
```

Q 7.3 (3 pts) Classe `FourmiFlottante`
La fourmi flottante est une fourmi dont la méthode `agir` consiste à lire les traces de la case où elle se trouve dans le monde et à modifier son opinion en conséquence. Donner le code de cette classe.

```
1 public class FourmiFlottante extends Fourmi{
2
3     public FourmiFlottante(double[] position, double opinion) {
4         super(position, opinion);
5     }
6
7     public void agir(Monde m) {
8         boolean trace = m.getTrace(getPosition());
9         this.modifierOpinion(trace);
10    }
11 }
```

Q 7.4 (4 pts) Fourmis actives
Les fourmis actives ont en commun de laisser des traces de leur opinion sur le terrain. Dans le détail, les **évangélisatrices** laisse la trace de leur opinion tandis que les **contradicteurs** sondent l'opinion du monde et prennent l'opinion opposée à la majorité avant de laisser leur trace.
Donner le code de la méthode `agir` pour toutes les classes restantes (en accord avec l'architecture de la Q7.1). Vous préciserez explicitement les classes où cette méthode n'est pas redéfinie. Donner aussi les constructeurs de ces 3 classes. Eviter toute duplication de code.

```
1 public abstract class FourmiActive extends Fourmi{
2     public FourmiActive(double[] position, double opinion) {
3         super(position, opinion);
4     }
5     public void agir(Monde m){ // toutes les fourmis actives laissent des traces
6         m.setTrace(getPosition(), this.voter()); // laisse une trace
              correspondant à son vote
7     }
8 }
9
10 public class FourmiEvangeliste extends FourmiActive{
11     public FourmiEvangeliste(double[] position, double opinion) {
12         super(position, opinion);
13     }
14     // Pas besoin de agir (c'est dans la classe mère)
15 }
16
17 public class FourmiContradictoire extends FourmiActive{
18     public FourmiContradictoire(double[] position) { // NE PAS demander d'opinion
           à la création (pas de sens)
19         super(position, Math.random());
20     }
21     public void agir(Monde m) {
22         setOpinion(m.getSondage()[0]>m.getSondage()[1]?0:1); // modifier l'
               opinion de la fourmi en fonction du monde
23         super.agir(m); // appliquer la nouvelle opinion
24     }
25 }
```

Q 7.5 (4 pts) Fourmis évangélisatrices plus astucieuses
Afin d'éviter de tourner en rond à cause de déplacement aléatoire, nous souhaitons implémenter une nouvelle stratégie de déplacement pour les fourmis évangélisatrices seulement. Ces fourmis vont se déplacer toujours vers en bas à droite (`position[0]` et `position[1]` seront incrémentés avec des réels aléatoires tirés entre 0 et 1 exclu). L'hypothèse du monde torique devrait leur permettre de mieux couvrir le terrain.
L'attribut `position` reste privé mais nous introduisons une méthode `protected void bouger(double x,double y)` dans la classe mère `Fourmi` qui permet de déplacer la position de la fourmi de `x,y`.
Quel(s) code(s) faut-il ajouter à quel(s) endroit(s) pour que les fourmis évangélisatrices se comportent comme nous le souhaitons ? Faut-il modifier dans la méthode `void update()` de la classe `Monde` ?

```
1 // Dans fourmi
2     protected void bouger(double x, double y){ // necessaire pour la Q7.5 mais
          pas avant
3         position[0] += x; position[1] += y;
4         position[0] = (position[0] + Monde.DIM) % Monde.DIM; // monde torique
5         position[1] = (position[1] + Monde.DIM) % Monde.DIM;
6     }
7 // dans évangélisatrice
8     protected void bouger(){ // surcharge
9       bouger(Math.random(), Math.random());
10    }
```

Exercice 8 – Simulation (4 pts)

Le but du code créé est de prédire le vote de la population à partir de différentes configurations. Nous allons créer une classe de test nommée `Simulation` et contenant un `main`. Dans cette simulation nous souhaitons :
– créer la population de 20 fourmis (avec des positions aléatoires) : 75% de flottantes, 10% d'évangélisatrices *pour*, 10% évangélisatrices *contre* et 5% de contradicteurs.
– créer un monde à partir de cette population et le faire évoluer sur 200 itérations,
– afficher le vote de la population.
Donner le code de cette classe.

```
1 public static void main(String[] args) {
2
3         Fourmi[] pop = new Fourmi[20];
4         int i;
5         for(i=0; i<15; i++)
6             pop[i] = new FourmiFlottante(new double[]{Math.random()*Monde.DIM,
                  Math.random()*Monde.DIM}, Math.random());
7         pop[i++] = new FourmiEvangeliste(new double[]{Math.random()*Monde.DIM,
              Math.random()*Monde.DIM}, 0.);
8         pop[i++] = new FourmiEvangeliste(new double[]{Math.random()*Monde.DIM,
              Math.random()*Monde.DIM}, 0.);
9         pop[i++] = new FourmiEvangeliste(new double[]{Math.random()*Monde.DIM,
              Math.random()*Monde.DIM}, 1.);
10        pop[i++] = new FourmiEvangeliste(new double[]{Math.random()*Monde.DIM,
              Math.random()*Monde.DIM}, 1.);
11        pop[i++] = new FourmiContradicteur(new double[]{Math.random()*Monde.DIM,
              Math.random()*Monde.DIM});
```

```
12
13          int nIter = 200;
14          Monde monde = new Monde(pop);
15
16          for(i=0; i<nIter; i++){
17              if(i%10 == 0){// NON DEMANDE
18                  System.out.println("Iter: "+i +" " + monde.getSondage()[0]+" "+
                        monde.getSondage()[1]);
19                  System.out.println(monde); // NON DEMANDE
20              }
21              monde.update();
22          }
23          System.out.println("Iter: "+i +" " + monde.getSondage()[0]+" "+monde.
                getSondage()[1]);
24
25          ImageIO.write(monde.toImage(20), "png", new File("monde.png")); // NON
                DEMANDE
26      }
```