Programmation Java @ Et3
Polytech Paris-Saclay | 2024-25

Nom : BRACQ
Prénom : Paolo

Exercice 1

Vous allez développer une gestion simple d'autotamponneuses. Celles-ci sont caractérisées par :

— le fait qu'elles soient placées ou non sur une piste d'autotamponneuses, leur position étant alors représentée par une paire de valeurs réelles (coordonnées dans un plan à deux dimensions)

— le fait qu'elles soient occupées ou non, et si oui par quelle personne (il faut qu'elles soient placées pour pouvoir être occupées)

— le fait qu'elle soient allumées ou non (il faut qu'elles soient occupées pour pouvoir être allumées)

— le fait qu'elle soient clignotantes ou non (il faut qu'elles soient allumées pour pouvoir être cignotantes)

1. Créez une nouvelle classe `Autotamponneuse` dans un package `com.feteforraine`. Ajoutez les données nécessaires à la classe en choisissant les niveaux de visibilité appropriés.

#1.1.1 classe `Autotamponneuse`

```
package com.feteforraine;
/*classe représenatant une autotamponneuse*/
public class Autotamponneuse {
    private boolean onpist;
    private boolean occuped;
    private boolean turnon;
    private boolean light;
}
```

2. Ajoutez deux constructeurs à la classe : l'un sans paramètres, l'autre avec une coordonnée x et une coordonnée y. Faites en sorte de réutiliser ce qui peut l'être dans vos constructeurs.

#1.2.1 constructeur sans paramètres

```
public Autotamponneuse_0param(){
    onpist=false;
    occuped=false;
    turnon=false;
    light=false;
}
```

#1.2.2 constructeur avec 2 paramètres

```
public Autotamponneuse(int position_x, int position_y){
    this();
    onpist=true;
    x=position_x;
    y=position_y;
}
```

#1.2.3 Pour quelles raisons aurait-on intérêt à réutiliser (invoquer) d'autres constructeurs lorsqu'on définit des constructeurs ?

Afin de réduire la redondance du code et une maintenance de la facilité du code et une meilleure organisation du code.

3. Ajoutez des méthodes d'instance dans la classe Autotamponneuse pour consulter l'état d'une autotamponneuse particulière : `estOccupee`, `getNomOccupant`, `estAllumee`, `estClignotante`.

#1.3.1 méthodes demandées

```
/*Méthode consultation occupation*/
public estOccupee(){
    return occuped;
}
/*Méthode consultation utilisateur*/
public getNomOccupant(){
    if (occuped) {
        return user;
    }
    return "";
}
/*Méthode consultation allumage*/
public estAllumee(){
    return turnon;
}
/*Méthode consultation clignotement*/
public estClignotante(){
    return light;
}
```

#1.3.2 Pourquoi parle-t-on de "méthodes d'instances" ?

On parle de methodes d'instances car ce sont des fonctions définies dans une classe qui opèrent sur des instances (objets) de cette classe.

#1.3.3 Pourquoi ces méthodes ne contournent-elles pas le principe d'encapsulation des données ?

Elle ne contournent pas le principe d'encapsulation car ells sont présentes dans la classe .

4. Ajoutez une méthode `main` dans la classe `Autotamponneuse` afin de pouvoir définir un programme principal de test. Testez-y sur des exemples le code précédemment défini, et utilisez-la pour tester le code des questions suivantes.

#1.4.1 méthode `main`

```
public static void main(String[] args){
    /*instantiation d'un objet de la classe Autotamponneuse*/
    Autotamponneuse Voiture1 =new Autotamponneuse(5,6);
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println("- Occupée : "+ Voiture1.estOccupee());
    System.out.println(" - Occupée par : " + Voiture1.getNomOccupant());
    System.out.println(" - Est allumée : " +Voiture1.estAllumee());
    System.out.println(" - Est clignotante : " +Voiture1.estClignotante());
}
```

#1.4.2 affichage produit par l'exécution de votre méthode `main`

```
Voici les infos sur Voiture1 :
- Occupée : false
 - Occupée par : Nobody
 - Est allumée : false
 - Est clignotante : false

Process finished with exit code 0
```

#1.4.3 méthode `main` dans une nouvelle classe `TestAutotamponneuse` dans le package `com.feteforraine`

```
public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(5,6);
        System.out.println("Voici les infos sur Voiture1 :");
        System.out.println("- Occupée : "+ Voiture1.estOccupee());
        System.out.println(" - Occupée par : " + Voiture1.getNomOccupant());
        System.out.println(" - Est allumée : " +Voiture1.estAllumee());
        System.out.println(" - Est clignotante : " +Voiture1.estClignotante());
    }
}
```

#1.4.4 Comment faire en sorte d'exécuter la méthode `main` de la classe `com.feteforraine.TestAutotamponneuse` ?

```
Compiler les classes du package com.feteforraine puis exécuter la classe contenant la méthode main
```

#1.4.5 Pourquoi la méthode `main` est-elle `static` ? A quoi correspond son paramètre ?

```
Car elle s'applique en dehors des objets
```

#1.4.6 Quelles critiques pourriez-vous faire à l'approche consistant à tester le fonctionnement d'une classe au travers d'un programme de test exécuté par sa méthode `main` ? Quelles autres approches pourrait-on imaginer pour faire cela ?

```
Manque de structure :
Les tests écrits dans la méthode main ne suivent pas une structure standardisée, ce qui peut rendre le code difficile à lire et à maintenir.

Difficulté à exécuter des tests spécifiques :
Il peut être difficile d'exécuter des tests spécifiques sans modifier le code de la méthode main.

Mon idée d'autres approches est de créer une classe test.
```

5. Ajoutez le code nécessaire pour qu'une autotamponneuse ait désormais un identifiant entier unique. Cette valeur ne devra pas être choisie par l'utilisateur, et devra commencer à la valeur 1 pour la première autotamponneuse.

#1.5.1 code ajouté

```
private int Id;
private static int compteur=0;
```

et

```
public Autotamponneuse(){
    Id=++compteur;
    onpist=false;
    occuped=false;
    turnon=false;
    light=false;
}
```

et

```
/*Méthode consultation l'id*/
public int getID(){
    return Id;
}
```

#1.5.2 test du fonctionnement dans la méthode `main`

```
Voici les infos sur Voiture1 :
- Identifiant : 1
- Occupée : false
 - Occupée par : Nobody
```

```
 - Est allumée : false
 - Est clignotante : false
Voici les infos sur Voiture2 :
- Identifiant : 2
- Occupée : false
 - Occupée par : Nobody
 - Est allumée : false
 - Est clignotante : false
```

6. Redéfinissez dans la classe Autotamponneuse la méthode particulière suivante de la classe `java.lang.Object:String toString()` qui retourne une représentation sous forme de chaîne de caractères de l'état d'un objet. Des exemples de chaînes pourraient être :

```
[1] (5.0,5.0) libre éteinte non clignotante
[2] (7.0,2.0) occupée (Charles Darwin) éteinte non clignotante
[2] (5.1,5.0) occupée (Charles Darwin) allumée clignotante
```

#1.6.1 redéfinition de la méthode `toString`

```
@Override
public String toString(){
    return "["+Id+"] ("+x+"," + y+")"+
    (occuped ?"occupée (" + user +") ":" libre")+
    (turnon?" éteinte ":" allumée ") +
    (light?" clignotante":" non clignotante");
}
```

#1.6.2 test du fonctionnement dans la méthode `main`

```
Voici les infos sur Voiture1 :
[1] (5.0,6.0) libre allumée  non clignotante
Voici les infos sur Voiture2 :
[2] (8.0,6.0) libre allumée  non clignotante

Process finished with exit code 0
```

7. Ajoutez à présent des méthodes d'instance dans la classe `Autotamponneuse` pour modifier l'état d'une autotamponneuse particulière, en respectant les règles énoncées ci-dessus : `place`, `ajouteOccupant`, `enleveOccupant`, `allume`, `eteint`, `demarreClignotement`, `arreteClignotement`. Afin d'informer le programme appelant sur la modification effective ou non de l'objet correspondant, faites retourner à vos méthodes une valeur booléenne. On décide que ces méthodes ne pourront être accessibles qu'aux autres classes du même package (`com.feteforraine`).

#1.7.1 définition des méthodes

```
boolean place(float positionx, float positiony){
    if (onpist){
        x=positionx;
```

```
        y=positiony;
    }else{
        x=-1;
        y=-1;
    }
    return true;
}
```

et

```
boolean ajouteOccupant(String nom){
    if (onpist & !occuped ){
        user=nom;
        occuped=true;
        return true;
    }else{
        return false;
    }
}
```

et

```
boolean enleveOccupant(){
    if (onpist & occuped ){
        user="Nobody";
        occuped=false;
        return true;
    }else {
        return false;
    }
}
```

et

```
boolean allume (){
    if (occuped & !turnon ){
        turnon=true;
        return true;
    }else{
        return false;
    }
}
```

et

```
boolean eteint(){
    if(turnon){
        turnon=false;
        return true;
    }else{
        return false;
    }
}
```

et

```
boolean demarreClignotement(){
    if (turnon & !light){
        light=true;
        return true;
    }else{
        return false;
    }
}
```

et

```
boolean arreteClignotement(){
    if (light){
        light=false;
        return true;
    }else{
        return false;
```

```
    }
}
```

#1.7.2 test du fonctionnement dans la méthode `main` et trace d'exécution

```
public static void main(String[] args){
    /*instantiation d'un objet de la classe Autotamponneuse*/

    Autotamponneuse Voiture1 =new Autotamponneuse(5,6);
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.place(9,0);
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.ajouteOccupant("Moi");
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.allume();
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.demarreClignotement();
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.arreteClignotement();
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.eteint();
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
    Voiture1.enleveOccupant();
    System.out.println("Voici les infos sur Voiture1 :");
    System.out.println(Voiture1.toString());
}
```

```
Et le résultat:

Voici les infos sur Voiture1 :
[1] (5.0,6.0) libre éteinte  non clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0) libre éteinte  non clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0)occupée (Moi)  éteinte  non clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0)occupée (Moi)  allumée  non clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0)occupée (Moi)  allumée  clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0)occupée (Moi)  allumée  non clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0)occupée (Moi)  éteinte  non clignotante
Voici les infos sur Voiture1 :
[1] (9.0,0.0) libre éteinte  non clignotante

Process finished with exit code 0
```

#1.7.3 En créant une classe Test dans le *package par* défaut, invoquez la méthode `ajouteOccupant`. Que se passe-t-il, pourquoi ?

Cela ne fonctionne pas : *java: ajouteOccupant(java.lang.String) is not public in com.feteforraine.Autotamponneuse; cannot be accessed from outside package.*
Et cela est tout à fait normal car lors de la définition de nos methodes nous n'avons pas précisé la visibilité, autrement dit seulement le package dans lequel elles sont définies peut les appeler. Or nous avons créée la classe T est dans un autre package, les methodes sont donc inutilisables.

8. On souhaite détecter des collisions entre autotamponneuses. Pour simplifier, on dira que deux autotamponneuses sont en collision si la distance entre leurs positions (comprise ici comme un point dans le plan) est inférieure à une valeur constante définie au niveau de la classe, `DISTANCE_MINIMALE`. Ajoutez cette valeur de manière appropriée, puis ajoutez une méthode d'instance pour le calcul de distance entre deux autotamponneuses :

`double calculeDistance(Autotamponneuse autreAuto)`

ainsi qu'une méthode d'instance indiquant si une collision a lieu entre deux autotamponneuses :

`boolean collision(Autotamponneuse autreAuto)`

#1.8.1 déclaration du champ `DISTANCE MINIMALE`

```
public static final double DISTANCE_MINIMALE = 1.00;
```

#1.8.2 méthode `calculeDistance`

```
private double calculeDistance(Autotamponneuse autreAuto){
    double X=x-autreAuto.x;
    double Y= y-autreAuto.y;
    return Math.sqrt(Math.pow(X,2)+Math.pow(Y,2));
}
```

#1.8.3 méthode `collision`

```
boolean collision(Autotamponneuse autreAuto){
    return calculeDistance(autreAuto)<=DISTANCE_MINIMALE;
}
```

#1.8.4 exemples de tests de collisions dans la classe `TestAutotamponneuse`

```
package com.feteforraine;

public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(1,2);
        com.feteforraine.Autotamponneuse Voiture2 =new com.feteforraine.Autotam-
ponneuse(5,6);
        System.out.println("Voici les infos sur la colisione entre la Voiture 1
et la Voiture 2 :");
        System.out.println("- Collision : "+ Voiture1.collision(Voiture2));
```

```
            }
}
```

```
Voici les infos sur la colisione entre la Voiture 1 et la Voiture
2 :
- Collision : false

Process finished with exit code 0
```

```
package com.feteforraine;

public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(5,5);
        com.feteforraine.Autotamponneuse Voiture2 =new com.feteforraine.Autotam-
ponneuse(5,6);
        System.out.println("Voici les infos sur la colisione entre la Voiture 1
et la Voiture 2 :");
        System.out.println("- Collision : "+ Voiture1.collision(Voiture2));
            }
}
```

```
Voici les infos sur la colisione entre la Voiture 1 et la Voiture
2 :
- Collision : true

Process finished with exit code 0
```

9. Afin d'autoriser d'autres types d'utilisation, proposez des équivalents sous forme de méthodes de classes pour les deux méthodes d'instance précédentes :

`static double calculeDistance(Autotamponneuse auto1, Autotamponneuse auto2)`

et :

`static boolean collision(Autotamponneuse auto1, Autotamponneuse auto2)`

Bien prendre en compte le fait que les paramètres peuvent tous les deux avoir une valeur `null` à l'exécution.

#1.9.1 méthode de classes `calculeDistance`

```
static double calculeDistance(Autotamponneuse auto1, Autotamponneuse auto2) {
    if (auto1 != null) {
        return auto1.calculeDistance(auto2);
    }else{
        return -1; /*J'ai choisi -1 car java demande un deuxième élément*/
    }
}
```

#1.9.2 méthode de classes `collision`

```
static boolean collision(Autotamponneuse auto1, Autotamponneuse auto2) {
    if (auto1 != null) {
        return auto1.collision(auto2);
    }else{
        return false;/*J'ai choisi false car java demande un deuxième élément*/
    }
}
static boolean collision(Autotamponneuse auto1, Autotamponneuse auto2) {
    if (auto1 != null) {
        return auto1.collision(auto2);
    }else{
        return false;/*J'ai choisi false car java demande un deuxième élément*/
    }
}
```

#1.9.3 exemples de tests de collisions dans la classe `TestAutotamponneuse`

```
package com.feteforraine;

public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(1,2);
        com.feteforraine.Autotamponneuse Voiture2 =new com.feteforraine.Autotam-
ponneuse(5,6);
        System.out.println("Voici les infos sur la colisione entre la Voiture 1
et la Voiture 2 :");
        System.out.println("- Distance : "+ com.feteforraine.Autotampon-
neuse.calculeDistance(Voiture1,Voiture2));
        System.out.println("- Collision : "+ com.feteforraine.Autotampon-
neuse.collision(Voiture1,Voiture2));
    }
}
```

```
Voici les infos sur la colisione entre la Voiture 1 et la Voiture
2 :
- Distance : 5.656854249492381
- Collision : false

Process finished with exit code 0
```

```
package com.feteforraine;

public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(6,6);
        com.feteforraine.Autotamponneuse Voiture2 =new com.feteforraine.Autotam-
ponneuse(5,6);
        System.out.println("Voici les infos sur la colisione entre la Voiture 1
```

```
et la Voiture 2 :");
        System.out.println("- Distance : "+ com.feteforraine.Autotampon-
neuse.calculeDistance(Voiture1,Voiture2));
        System.out.println("- Collision : "+ com.feteforraine.Autotampon-
neuse.collision(Voiture1,Voiture2));
    }
}
```

```
Voici les infos sur la colisione entre la Voiture 1 et la Voiture
2 :
- Distance : 1.0
- Collision : true

Process finished with exit code 0
```

#1.9.4 La définition des méthodes de classes pourrait-elle réutiliser celle de la méthode d'instances correspondante ? Si oui, pourquoi ?

```
On peut effectivement réutiliser la méthode dinstance afin
d'éviter la redondance et clarifier le code
```

10. Redéfinissez dans la classe `Autotamponneuse` la méthode de la classe `java.lang.Object` :

```
boolean equals(Object autreObjet)
```

qui indique si deux objets sont égaux (même état complet ici). Attention, la méthode étant déclarée au niveau de la classe `java.lang.Object`, il faudra vérifier puis transtyper le type du paramètre de la méthode (`Object autreObjet`).

#1.10.1 redéfinition de la méthode `equals`

```
@Override
public boolean equals(Object autreObjet){
    Autotamponneuse auto =  (Autotamponneuse) autreObjet;
    if (onpist != auto.onpist){
        return false;
    }
    if (turnon != auto.turnon){
        return false;
    }
    if (occuped != auto.occuped){
        return false;
    }
    if (light != auto.light){
        return false;
    }
    if (x != auto.x){
        return false;
    }
    if (y != auto.y){
        return false;
    }
    if (Id != auto.Id){
```

```
        return false;
    }
    if (user != auto.user){
        return false;
    }
    return true;
}
```

#1.10.2 tests d'égalité entre objets dans la classe `TestAutotamponneuse`

```
package com.feteforraine;

public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(6,6);
        Voiture1.ajouteOccupant("Moi");
        Voiture1.allume();
        Voiture1.demarreClignotement();
        com.feteforraine.Autotamponneuse Voiture2 =new com.feteforraine.Autotam-
ponneuse(6,6);
        Voiture2.ajouteOccupant("Moi");
        Voiture2.allume();
        Voiture2.demarreClignotement();
        System.out.println("Voici les infos sur l'égalité entre la Voiture 1 et
la Voiture 2 :");
        System.out.println(" - Egales : "+ Voiture1.equals(Voiture2));
    }
}
```

```
Voici les infos sur l'égalité entre la Voiture 1 et la Voiture 2 :
 - Egales : true

Process finished with exit code 0
```

```
package com.feteforraine;

public class TestAutotamponneuse {
    public static void main(String[] args){
        /*instantiation d'un objet de la classe Autotamponneuse*/
        com.feteforraine.Autotamponneuse Voiture1 =new com.feteforraine.Autotam-
ponneuse(1,3);
        Voiture1.ajouteOccupant("Toi");
        Voiture1.allume();
        Voiture1.demarreClignotement();
        com.feteforraine.Autotamponneuse Voiture2 =new com.feteforraine.Autotam-
ponneuse(6,6);
        Voiture2.ajouteOccupant("Moi");
        Voiture2.allume();
        Voiture2.demarreClignotement();
        System.out.println("Voici les infos sur l'égalité entre la Voiture 1 et
la Voiture 2 :");
        System.out.println(" - Egales : "+ Voiture1.equals(Voiture2));
```

```
    }
}
```

```
Voici les infos sur l'égalité entre la Voiture 1 et la Voiture 2 :
 - Egales : false

Process finished with exit code 0
```

11. Ajoutez une nouvelle classe `PisteAutotamponneuses` dans le package `com.feteforraine`. Une telle classe comporte une collection d'autotamponneuses représentée par un tableau. Ajoutez-y un constructeur prenant pour paramètre la taille de cette collection.

#1.11.1 définition de la classe `PisteAutotamponneuses`

```
public class PisteAutotamponneuses {
    private Autotamponneuse[] Voitures;
    private int size;

    public PisteAutotamponneuses(int taille) {
        // On initialise le tableau avec la taille spécifiée
        this.Voitures = new Autotamponneuse[taille];
        size=taille;
        for (int i = 0; i < taille; i++) {
            Voitures[i]=new com.feteforraine.Autotamponneuse();
            Voitures[i].miseEnPiste();
        }
    }
```

12. Ajoutez une nouvelle méthode statique `main` dans la classe `PisteAutotamponneuses` pour définir un programme de test qui crée le nombre d'autotamponneuses requis et les place de façon aléatoire sur la piste (cf. `java.lang.Math.random()`). Faites en sorte qu'aucune autotamponneuse nouvellement placée ne soit en collision avec une autre autotamponneuse.

#1.12.1 méthode `main`

```
public static void main(String[] args){
    int nbr= 20;
    com.feteforraine.PisteAutotamponneuses Piste1= new PisteAutotampon-
neuses(nbr);
    System.out.println("Voici les infos sur Piste1 :");
    for (int i = 0; i < nbr; i++) {
        float x= (float)Math.random()*10;
        float y= (float)Math.random()*10;
        boolean estPlace= Piste1.Voitures[i].place(x,y);
        for (int j = 0; j < i; j++) {
            while (com.feteforraine.Autotamponneuse.collision(Piste1.Voi-
tures[i],Piste1.Voitures[j])){
                x= (float)Math.random()*10;
                y= (float)Math.random()*10;
                estPlace= Piste1.Voitures[i].place(x,y);
            }
        }
```

```
            System.out.println(Piste1.Voitures[i].toString());
    }
}
```

#1.12.2 trace d'exécution de tests de création de piste

```
Voici les infos sur Piste1 :
[1] (2.14339,8.714767) libre éteinte  non clignotante
[2] (3.2952433,4.768322) libre éteinte  non clignotante
[3] (5.1470113,3.6282313) libre éteinte  non clignotante
[4] (3.1530359,0.7003111) libre éteinte  non clignotante
[5] (5.563969,8.554636) libre éteinte  non clignotante
[6] (3.7571466,5.7116847) libre éteinte  non clignotante
[7] (8.516029,2.2217546) libre éteinte  non clignotante
[8] (8.269333,8.250477) libre éteinte  non clignotante
[9] (6.8927197,2.8286636) libre éteinte  non clignotante
[10] (1.5961976,3.8478806) libre éteinte  non clignotante
[11] (7.7015977,0.65375423) libre éteinte  non clignotante
[12] (1.8980001,1.6475593) libre éteinte  non clignotante
[13] (9.996858,8.936836) libre éteinte  non clignotante
[14] (7.255898,3.7904856) libre éteinte  non clignotante
[15] (0.2729172,1.5485734) libre éteinte  non clignotante
[16] (8.038613,8.560059) libre éteinte  non clignotante
[17] (8.528797,5.955338) libre éteinte  non clignotante
[18] (1.3487973,5.4165707) libre éteinte  non clignotante
[19] (1.6802046,9.76956) libre éteinte  non clignotante
[20] (5.32116,0.7561294) libre éteinte  non clignotante

Process finished with exit code 0
```

13. Redéfinissez la méthode `String toString()` de la classe `PisteAutotamponneuses` afin qu'elle affiche l'état de la collection complète (l'ordre des autotamponneuses sera par ordre d'ajout). Utilisez pour créer la chaîne de caractères résultat la classe `java.lang.StringBuilder`.

#1.13.1 méthode `toString`

```
@Override
public String toString(){
    String result="";
    for (int i = 0; i < size; i++) {
        result+="Position liste : "+i+" Autotamponneuse : "+Voitures[i].ge-
tID()+" Place : ("+Voitures[i].getX()+"," + Voitures[i].getY()+") \n";
    }
    return result;
}
```

#1.13.2 test d'utilisation de la méthode `toString`

```
public static void main(String[] args){
        int nbr= 20;
```

```
        com.feteforraine.PisteAutotamponneuses Piste1= new PisteAutotampon-
neuses(nbr);
        for (int i = 0; i < nbr; i++) {
            float x= (float)Math.random()*10;
            float y= (float)Math.random()*10;
            boolean estPlace= Piste1.Voitures[i].place(x,y);
            for (int j = 0; j < i; j++) {
                while (com.feteforraine.Autotamponneuse.collision(Piste1.Voi-
tures[i],Piste1.Voitures[j])){
                    x= (float)Math.random()*10;
                    y= (float)Math.random()*10;
                    estPlace= Piste1.Voitures[i].place(x,y);
                }
            }
        }
        System.out.println("Voici les infos sur Piste1 :");
        System.out.println(Piste1.toString());

    }
}
```

```
Voici les infos sur Piste1 :
Position liste : 0 Autotamponneuse : 1 Place :
(6.2663727,4.5351167)
Position liste : 1 Autotamponneuse : 2 Place :
(1.8014379,1.2522485)
Position liste : 2 Autotamponneuse : 3 Place :
(0.034493536,3.2922742)
Position liste : 3 Autotamponneuse : 4 Place :
(6.4243026,1.2210363)
Position liste : 4 Autotamponneuse : 5 Place : (1.75211,4.8467393)
Position liste : 5 Autotamponneuse : 6 Place :
(3.992569,0.5440322)
Position liste : 6 Autotamponneuse : 7 Place :
(4.4050465,1.8371361)
Position liste : 7 Autotamponneuse : 8 Place :
(2.3983912,2.180089)
Position liste : 8 Autotamponneuse : 9 Place :
(4.1684275,8.008901)
Position liste : 9 Autotamponneuse : 10 Place :
(9.401939,9.907583)
Position liste : 10 Autotamponneuse : 11 Place :
(9.8996315,3.8093505)
Position liste : 11 Autotamponneuse : 12 Place :
(5.3658943,2.6336308)
Position liste : 12 Autotamponneuse : 13 Place :
(9.073514,0.63304454)
Position liste : 13 Autotamponneuse : 14 Place :
(9.021727,8.392566)
Position liste : 14 Autotamponneuse : 15 Place :
(5.606984,0.19235766)
Position liste : 15 Autotamponneuse : 16 Place :
(1.9150224,8.029111)
Position liste : 16 Autotamponneuse : 17 Place :
(7.3735285,7.6073117)
```

```
Position liste : 17 Autotamponneuse : 18 Place :
(6.3743105,4.240368)
Position liste : 18 Autotamponneuse : 19 Place :
(2.510948,9.046017)
Position liste : 19 Autotamponneuse : 20 Place :
(2.6961434,5.261635)


Process finished with exit code 0
```

14. (*facultatif*) Ajoutez une méthode `dereglementAleatoire` à la classe `PisteAutotamponneuses` qui déplace de façon aléatoire et continue chaque autotamponneuse encore pilotée tour à tour, et élimine les autotamponneuses entrées en collision. Cette méthode affichera l'historique des autotamponneuses éliminées, et le vainqueur (autotamponneuse survivante si elle existe).

#1.14.1 méthode `dereglementAleatoire`

```
...
```