# PROJET SESSION HIVER 2023-1-INF1573-01 PROGRAMMATION II

## THEME: Apports de la programmation par événements dans un environnement objet (Cas d'une Ville avec parking intelligent)


Présenté par:

**Groupe 4:**

- **Romaric Hyacinthe Sieyamdji. D**
- **Ndamen Fomen Japha Rhodian**
- **Noula Kamtchi Dave Collins**
- **Tchakounte Emeric Gaetan**

Supervisé par:

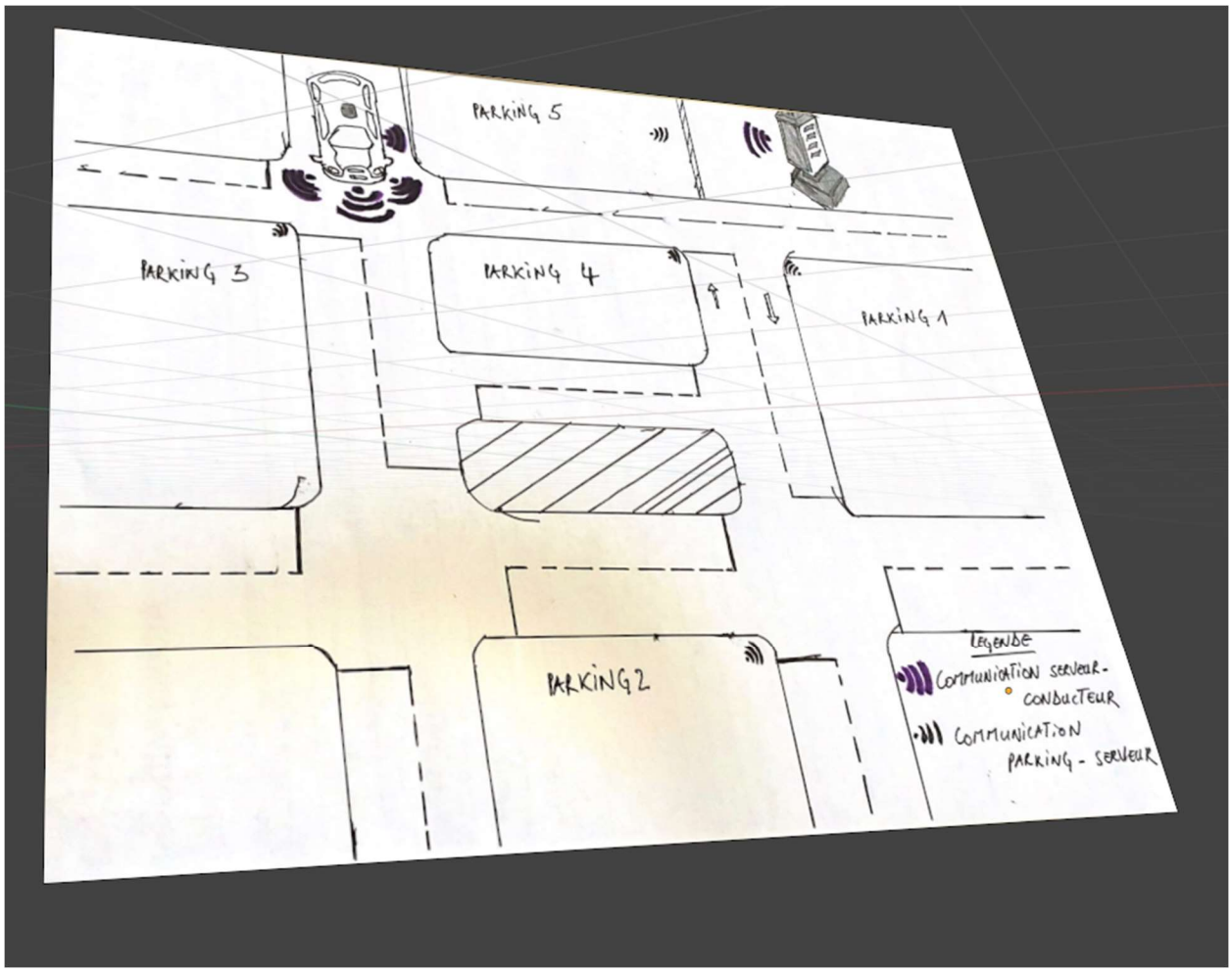
**Prof. Ilham Benyahia**

# Table de matières

# Introduction

À la suite de l'introduction à la programmation par évènement vu en cours de session, un travail de session a été donne portant sur ce sujet dans le cas d'une ville intelligente qui permet le parking facile. Pour parvenir à la mise en œuvre de cette application divers étape ont été suivi parmi lesquelles une analyse, une conception, une implémentation et des tests de qualité. Travail étant réalisé, ce rapport ci présent donnera un aperçu de la réalisation des différentes étapes précédemment citées

# 2. Analyse




Sur l'image ci-dessus nous avons une illustration de notre interface. Sur celle-ci nous pouvons voir nos cinq différents parkings chacun communiquant avec notre serveur, des routes et des intersections ensuite une illustration de la voiture qui communique avec le serveur pour avoir un parking à proximité dans l'espace de parkings à noter qu'il pourrait être n'importe où dans la zone!

# 3. Conception

# a. Interfaces (Composantes graphiques)

## Classes importées de l'API de JAVA pour la réalisation de notre projet

| Nom de la composante (Classe) | Rôle selon l'API de java | Méthodes associées | Descriptions dans le contexte du code fourni |
|---|---|---|---|
| **JFrame** | Classe de la bibliothèque graphique de java qui représente une fenêtre graphique. **Jframe** est utilisé pour créer des applications graphiques avec une interface utilisateur basée sur des fenêtre.<br><br>Nous l'avons utilisé pour l'affichage de la ville et l'interface ou nous devons faire entrer les données | **SetTitle (String title)** | Permet de définir le titre de la fenêtre (la ville que nous avons créée) |
| | | **setSize (int width, int height)** | Permet de définir la taille de la ville crée |
| | | **setDefaultCloseOperation () «int operation** | Permet de définir l'action qui doit être effectuée lorsque l'utilisateur ferme la fenêtre |
| | | **setVisible (Boolean b)** | Permet de rendre la fenêtre (ville) visible |
| | | **add (Component comp)** | Permet d'ajouter un composant à notre ville |
| | | **setLayout (LayoutManager manager)** | Permet de définir un gestionnaire de disposition utilisé pour positionner les composants dans la ville |
| **Graphics** | Classe permettant de fournir des méthodes pour dessiner (tracer des lignes, des formes, des textes et des images) des | **PaintComponent ()** | Permet de dessiner les différents éléments de notre ville graphique tels que les routes, les stations, le conducteur |

| | | | |
|---|---|---|---|
| | graphiques 2D dans une zone de dessin (cette zone de dessin peut être une fenêtre, un panneau …) | | |
| **Color** | Représente une couleur avec des constantes prédéfinies pour les couleurs | **SetColor ()** | Permet de définir la couleur de remplissage d'un composant créer en utilisant la constante prédéfinie<br>`**Color.nomDeLaCouleur**`<br>Nous avons utilisé cette méthode pour donner les couleurs des différents composants (les stations, le conducteur, les routes) de notre ville graphique |
| | | FillRect () | Permet de dessiner un rectangle |
| **Graphics2D** | Elle nous offre plus de possibilité dans la représentation de nos différents composant car Graphics seul ne suffit pas.<br><br>Est une classe de la bibliothèque graphique AWT en java qui étend de la classe Graphics. Elle offre des fonctionnalités pour les | **Paint (Graphics g)** | Permet de dessiner les différents éléments de notre ville graphique tels que les routes, les stations, le conducteur |
| | | **DrawLine ()** | Permet de tracer une ligne de la couleur donnée entre deux points. |
| | | **SetColor (Color. Nom de la couleur)** | Permet de définir la couleur d'un composant |
| | | | Permet de définir la largeur et le style de la bordure de dessin pour les formes dessinées |
| | | **Fill3DRect ()** | Permet de dessiner un rectangle en 3D avec une bordure. Elle prend en argument 5 arguments : la position x et y d'un coin supérieur gauche du rectangle, la largeur, la hauteur du rectangle et un booléen qui indique si le rectangle doit être enfoncé ou en relief |

| | | | |
|---|---|---|---|
| | dessins et les images | **DrawString ()** | Permet de dessiner du texte sur un composant graphique Swing tel qu'un JPanel. Cette méthode prend en argument une chaîne de caractères à dessiner et les cordonnées (x, y) où le texte doit être dessiné . |
| **ArrayList** | Classe permettant de stocker des objets dans une liste dynamique (c'est à dire que la taille peut changer pendant l'exécution du programme) | **Add (Object élément)** | Ajoute un élément spécifié la fin de la liste. **Syntaxe** :<br>**nomObject.add (élément)**<br>Utilisé dans notre programme pour le stockage et la manipulation de nos différents données |
| | | **Get (Int index)** | Permet de retourner l'élément à l'index spécifié dans la liste |
| | | **Clear ()** | Permet de supprimer tous les éléments de la liste, la laissant vide<br>. **Syntaxe** :<br>nomObject.add () |
| **JOptionPane** | Classe permettant d'afficher des boîtes de dialogue modales avec des messages, des boutons et des champs de saisie. Elle fournit une interface graphique pour les interactions | **ShowMessageDialog ()** | Permet d'afficher une boite de dialogue avec un message.<br>**Syntaxe** :<br>**JOptionPane. ShowMessageDialog** (null, `votre message ici`)<br>Utilisé dans notre projet pour nous renvoyer les messages des exceptions rencontrer (EX : le conducteur doit être sur la route pour pouvoir demander un stationnement , le conducteur ne doit pas être dans un stationnement et faire une demande stationnement …) |

| | | | |
|---|---|---|---|
| | avec l'utilisateur | **ShowInputDialog ()** | Affiche une boîte de dialogue avec une invite de saisie pour l'utilisateur.<br>Utilisé dans notre programme pour pouvoir récupérer les coordonnés (X et Y) du conducteur |
| **JPanel** | Permet de créer des conteneurs pour des composants graphiques tels que des boutons, des champs de texte, des étiquettes. | **Add ()** | Permet d'ajouter un bouton à panneau, permet d'ajouter nos différents composants (route, stations, conducteur) et les textes Arrays dans l'interface de saisis des coordonnées du conducteur |
| | | **SetSize ()** | Permet de définir la taille de notre panneau |
| **JButton** | Classe qui permet de créer des boutons interactifs dans une interface utilisateur | **JButton (String text)** | Le constructeur de la classe **JButton** qui permet de créer un nouveau bouton avec un texte spécifié<br>Utilisé pour créer un bouton, mettre l'activation et la validation de actionPerformed |
| | | **Add ActionListener (ActionListener listener)** | Permet d'ajouter un objet **ActionListener** au bouton, qui est appelé lorsque le bouton est cliqué |
| | | **SetPreferredSize (Dimension size)** | Permet de définir la taille préférée du bouton |
| **JLabel** | Permet de créer des composants graphiques pour afficher du texte ou des images dans une interface utilisateur | **JLabel (String text)** | Le constructeur de **JLabel** qui permet de créer une nouvelle étiquette avec un texte spécifié.<br>Nous l'avons utilisé pour écrire du texte sur notre interface (le nom des stations), afficher le message demandant au conducteur (sur l'interface) d' entrer ses coordonnés<br>Pour l'affichage de la station la plus proche |

| | | | |
|---|---|---|---|
| Interface :<br><br>**ActionListener** | Est une interface de java, qui permet de créer des évènements pour les composants graphiques qui déclenchent une action lorsqu'ils sont activés | **ActionPerformed (ActionEvent e)** | Cette méthode est appelée lorsque l ' action est déclenchée. L'objet **ActionEvent** passé en paramètre contient des informations sur l'évènement, comme l'objet source qui a déclenché l'action.<br><br>Utiliser pour surveiller l'état du bouton pour pouvoir activer actionPerformed |
| **ActionEvent** | Classe qui permet de représenter un évènement d'action déclenché par un utilisateur ou par le code de l'application | **GetSource ()** | Cette méthode renvoie l'objet qui a déclenché l'évènement |
| | | **GetID ()** | Cette méthode renvoie l'id de l'évènement |
| | | **ToString ()** | Renvoie une chaîne qui décrit l'événement |
| **BorderLayout** | Gestionnaire de disposition qui permet d'organiser les composants graphiques dans une fenêtre ou un panneau en utilisant cinq zones : nord, sud, est, ouest et centre<br><br>Utiliser pour bien définir l'encadrement des différent composant se notre interface graphique | | |
| **GirdLayout** | Classe de disposition qui permet d'organiser les composants graphiques dans. Elle prend en argument 2 paramètres (le nombre de lignes et le nombre de colonnes de la grille) | | |
| | | | |
| **BasicStroke** | Est une classe de la bibliothèque graphique java 2D qui permet de définir les propriétés des traits qui seront utilisés pour dessiner des formes, des lignes ou des contours | **SetStrocke ()** | Permet de définir la largeur et le style de la bordure de dessin pour les formes dessinées.<br><br>Cette méthode nous a permis d'augmenter la largeur des lignes que nous avons créé qui représente des lignes sur notre interface |

| | | | |
|---|---|---|---|
| **List** | Est une interface qui représente une collection ordonnée d'éléments | **Add ()** | Permet de stocker l'ensemble des nœuds adjacents à un nœud source pour la formation de notre graphe |

## b. Diagramme des classes UML.

Pour la réussite de notre conception le diagramme de classes a été choisi par rapport aux autres diagrammes d'UML car il permet de mettre efficacement en exergue tous nos acteurs(classes) et la communication qui se passent entre elles dans le but de répondre à besoin de parking pour éviter une congestion.

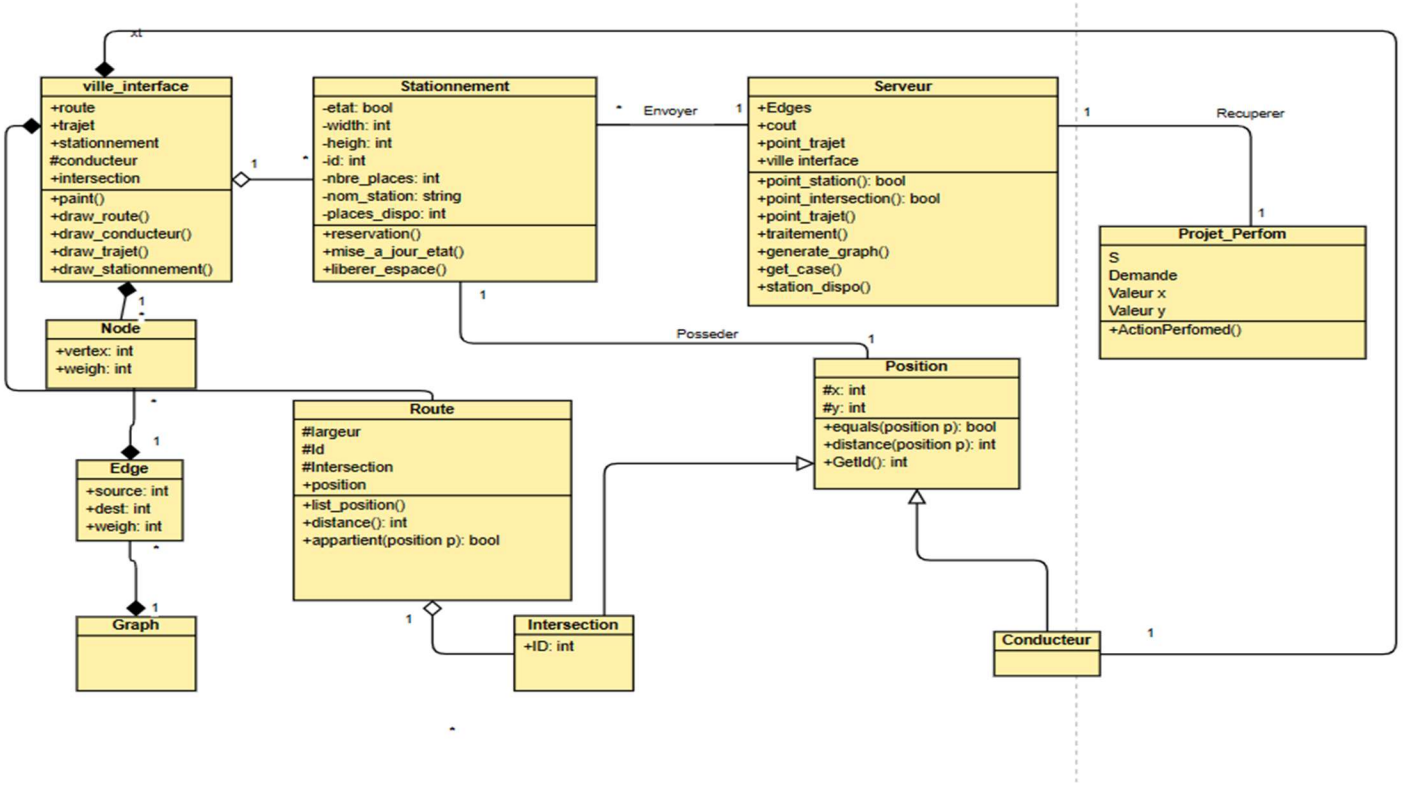
En première phase nous allons donner toutes les classes et leurs différentes descriptions chacune et par la suite nous allons les mettre en collaboration dans le diagramme de classes pour plus de clarté.

Le tableau ci-dessous contiendra toutes les classes de notre système :

| **Classes** | **Rôles** |
|---|---|
| Node | Représente les nœuds. |
| Edge | Forme une ligne et le poids entre les nœuds. |
| Graph | Permet de faire des graphes de la ville. |
| Intersection | Représente nos intersections. |
| Position | Représente une position donnée dans la ville |
| Stationnement | Représente nos stationnements. |

| | |
|---|---|
| Route | Représente les routes(association) |
| Serveur | Pour la réalisation du traitement en arrière-plan et fournir le trajet optimal |
| Conducteur | Représente le conducteur dans sa voiture |
| Test_Api | Pour tester |
| Projet_perfom | Pour rentrer les coordonnées et réaliser l'affichage après traitement. |
| Ville_Interface | Représente la ville |

Ensuite, la capture d'écran ci-dessous représente le diagramme de classes




## Règles d'association :

R1 : Une Edge est composée de plusieurs Nodes

R2 : Un graph est composé de plusieurs Edge

R3 : Une ville a plusieurs Nodes.

R4 : Une ville a plusieurs routes.

R5 : Une ville a au moins un conducteur.

R6 : Une ville a plusieurs stationnements.

R7 : Une route a une ou plusieurs intersections.

R8 : Un ou plusieurs stationnements envoient au serveur (transfert d'information).

R9 : Un stationnement possède une position

R10 : Un serveur récupère des données dans Projet_perfom.

**Explication brève:**

Lorsqu'un conducteur arrive dans la ville et cherche un stationnement il va sur l'interface de parking de la ville et entre sa position. Ensuite un serveur récupère les positions entrées à partir de la classe Projet_perfom puis il fait le traitement ayant déjà reçue(continuellement) toutes les informations sur les parkings

# 4. Implémentation

Cela étant, une conception réussie il est question de faire une implémentation ainsi la suite de cette rédaction est portée sur le codage. Pour se faire la concentration est portée sur des bouts de codes les plus importants de notre application.

## a. Bout de code classe Ville_Interface

Cette classe permet dessiner(matérialiser) l'interface de la ville avec les routes, trajets, conducteur…

```
public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2d=(Graphics2D) g;
    Drawroute(g2d,r1);
    Drawroute(g2d,r2);
    Drawroute(g2d,r3);
    Drawroute(g2d,r4);
    Drawroute(g2d,r5);
    Drawroute(g2d,r6);
    Drawroute(g2d,r7);
    Drawroute(g2d,r8);
    Drawroute(g2d,r9);
    Drawroute(g2d,r10);
    Drawroute(g2d,r11);
    Drawroute(g2d,r12);
    Drawroute(g2d,r13);
    Drawroute(g2d,r14);
    Drawstationnement(g2d,st1);
    Drawstationnement(g2d,st2);
    Drawstationnement(g2d,st3);
    Drawstationnement(g2d,st4);
    Drawstationnement(g2d,st5);
    Drawconducteur(g2d,cd);
    if(trajet.size()>1) {for(int i=1;i<trajet.size();i++)
    Drawtrajet(g2d,trajet.get(i),trajet.get(i-1) );}
    }
```

Cette méthode de la classe Graphics 2D permet de dessiner les différents composants de la ville

```
public void Drawroute(Graphics2D g,route r) {

    g.setColor(Color.black);
    g.setStroke(new BasicStroke(r.getLargeur()));
    g.drawLine(r.getP1().getX(),r.getP1().getY(),r.getP2().getX(),r.getP2().getY());
}
```

La méthode Drawroute qui prend en paramètre la route r et Graphics2D, elle permet de dessiner les routes, tracer les routes et les mettre les couleurs en noir comme couleur de fond.

```
public void Drawstationnement(Graphics2D g,Stationnement sta) {
    g.setColor(Color.GREEN);
    g.fill3DRect(sta.getPosition().getX(),sta.getPosition().getY(),sta.getWidth(), sta.getHeigth(),sta.getEtat());
    g.setColor(Color.black);
    g.drawString(sta.getNom_station(),sta.getPosition().getX()+40,sta.getPosition().getY()+60);
    g.drawString("nombre de place:"+sta.getNombre_de_place(),sta.getPosition().getX()+40,sta.getPosition().getY()+70);
    g.drawString("nombre de place disponibles:"+sta.getPlace_Disponibles(),sta.getPosition().getX()+40,sta.getPosition().getY()+80);
}
```

La méthode Drawstationnment comme présentée va permettre de dessiner nos différentes stations en les mettant des couleur une couleur verte clair pour station libre et vert sombre pour station indisponible.

```
public void Drawconducteur(Graphics2D g,conducteur a) {
    g.setColor(Color.RED);
    g.fill3DRect(a.getX(),a.getY(),20, 20,true);

}
```

Dans la même optique que les méthodes précédentes la méthode Drawconducteur va dessiner le conducteur et le représenter par un rectangle 3D en rouge.

```
public void Drawtrajet(Graphics2D g,Position p1,Position p2) {
    g.setColor(Color.BLUE);
    g.setStroke(new BasicStroke(10));
    g.drawLine(p1.getX(),p1.getY(),p2.getX(),p2.getY());
}
}
```

Cette méthode dessine le trajet en bleu après que le chemin a été calculer par le serveur.

### b. Bout de code classe stationnement

```
public void miseajouretat() {
    if(this.Nombre_de_place-this.getPlace_Disponibles()==this.Nombre_de_place)this.etat=false;
    else this.etat=true;
}
```

Cette méthode permet de faire la mise a jour de l'état en fonction des places_disponibles et du nombre_de_place.

```
    public void reservation() {
        if(etat) {this.Place_Disponibles--;
        System.out.println("reservation effectuÃ©e");}
        else System.out.println("plus de place");
        miseajouretat();
    }
```

Réservation () permet de réserver une place de stationnement.

```
    public void liberer_espace() {
        if(this.Place_Disponibles<this.Nombre_de_place)this.Place_Disponibles++;
        miseajouretat();
    }

}
```

La méthode libérer_espace () permet de libérer l'espace dans le stationnement.

## C. Bout code classe route

```
public int distance() {
    int distance=0;
    double x=Math.pow((p1.getX()-p2.getX()), 2);
    double y=Math.pow((p1.getY()-p2.getY()), 2);
    distance=(int) (Math.sqrt(x+y));
    return distance;
}
```

L'une des méthodes les plus importantes de classe route est la méthode distance qui après avoir récupérer deux points  elle calcule et retourne la valeur de leur distance.

```
69    public void listePosition() {
70        int min=0;int max=0;
71        if(p1.getX()==p2.getX()) {
72            for(int i=p1.getX()-(this.largeur/2);i<=p1.getX()+(this.largeur/2);i++) {
73                if(p1.getY()>p2.getY()) {
74                    min=p2.getY();max=p1.getY();
75                }
76                else {
77                    min=p1.getY();max=p2.getY();
78                }
79                for(int j=min;j<=max;j++) {
80                    position.add(new Position(i,j));
81                }
82            }
83        }
84        else if(p1.getY()==p2.getY()) {
85            for(int i=p1.getY()-(this.largeur/2);i<=p1.getY()+(this.largeur/2);i++) {
86                if(p1.getX()>p2.getX()) {
87                    min=p2.getX();max=p1.getX();
88                }
89                else {
90                    min=p1.getX();max=p2.getX();
91                }
92                for(int j=min;j<=max;j++) {
93                    position.add(new Position(j,i));
94                }
95            }
96        }
97    }
```

De façon brève cette méthode permet de générer un tableau constitué de toutes les positions dans une route.

```
103     public boolean appartient(Position p) {
104         boolean resultat=false;
105         for(Position s:this.position) {if(p.equals(s)) {resultat= true;
106      return resultat;}}
107         return resultat;
108     }
109 }
110
```

Cette méthode de type Boolean permet de savoir si un point appartient a une route et le résultat retourné sera true ou false.

## D. Bout de code classe Projet_perfom

```
54     public void actionPerformed(ActionEvent e) {
55         if(e.getSource()==demande) {
56             int X;int Y;
57             try {
58                 //recuperation de valeurs communiquÃ©es
59             X=Integer.parseInt(valeurX.getText());
60             Y=Integer.parseInt(valeurY.getText());
61             //initialisation des donnÃ©es du serveur
62             s=new serveur();
63             s.v.cd.setX(X);
64             s.v.cd.setY(Y);
65             //lancement du traitement
66             s.traitement(s.v.cd);
67             }catch(NumberFormatException e1) {JOptionPane.showMessageDialog(this, "le format rentrÃ© ne correspond pas aux entiers", "erreur d
68             catch(NullPointerException e2) {JOptionPane.showMessageDialog(this, "rentrez vos coordonnÃ©es", "erreur d'entree", JOptionPane.ERR
69             finally {System.out.println("fini");
70             //tracage du chemin
71             for(Position t:s.pointTrajet) {
72                 //recuperation des positions a parcourir
73                 s.v.trajet.add(t);
74                     repaint();
75                 }
76             }
77         }
78         }
79
80     }
```

Comme il est visible sur la capture d'écran le bout de code de cette méthode permet lancer un traitement après que l'user ait rentre ses coordonnées. Mais afin de gérer les mauvaises entrées des exceptions ont été prise en compte et un finally pour exécuter le code est conservé les données en cas d'erreur. Ensuite il y'a une boucle for qui parcoure le tableau dynamique des trajets et récupère ensuite les positions à parcourir après cela ajoute le trajet au serveur et Paint le chemin.

## E. Bout de code serveur

Ces deux extraits de code de la classe serveur évaluent le chemin partant du point aux différentes stations et retourne un tableau de points menant vers la station de plus petit itinéraire.

```java
public  int findShortestPaths(Graph graph, int source, int n)
{
    // crée un min-heap et pousse le nœud source ayant une distance de 0
    PriorityQueue<Node> minHeap;
    minHeap = new PriorityQueue<>(Comparator.comparingInt(node -> node.weight));
    minHeap.add(new Node(source, 0));

    // définit la distance initiale de la source à `v` comme infini
    List<Integer> dist;
    dist = new ArrayList<>(Collections.nCopies(n, Integer.MAX_VALUE));

    // la distance de la source à elle-même est nulle
    dist.set(source, 0);

    // array booléen pour suivre les sommets pour lesquels le minimum
    // le coût est déjà trouvé
    boolean[] done = new boolean[n];
    done[source] = true;

    // stocke le prédécesseur d'un sommet (dans un chemin d'impression)
    int[] prev = new int[n];
    prev[source] = -1;

    // exécuter jusqu'à ce que le Min-Heap soit vide
    while (!minHeap.isEmpty())
    {
        // Supprime et renvoie le meilleur sommet
        Node node = minHeap.poll();

        // récupère le numéro du sommet
        int u = node.vertex;;

        // faire pour chaque voisin `v` de `u`
        for (Edge edge: graph.adjList.get(u))
        {
            int v = edge.dest;
            int weight = edge.weight;
```

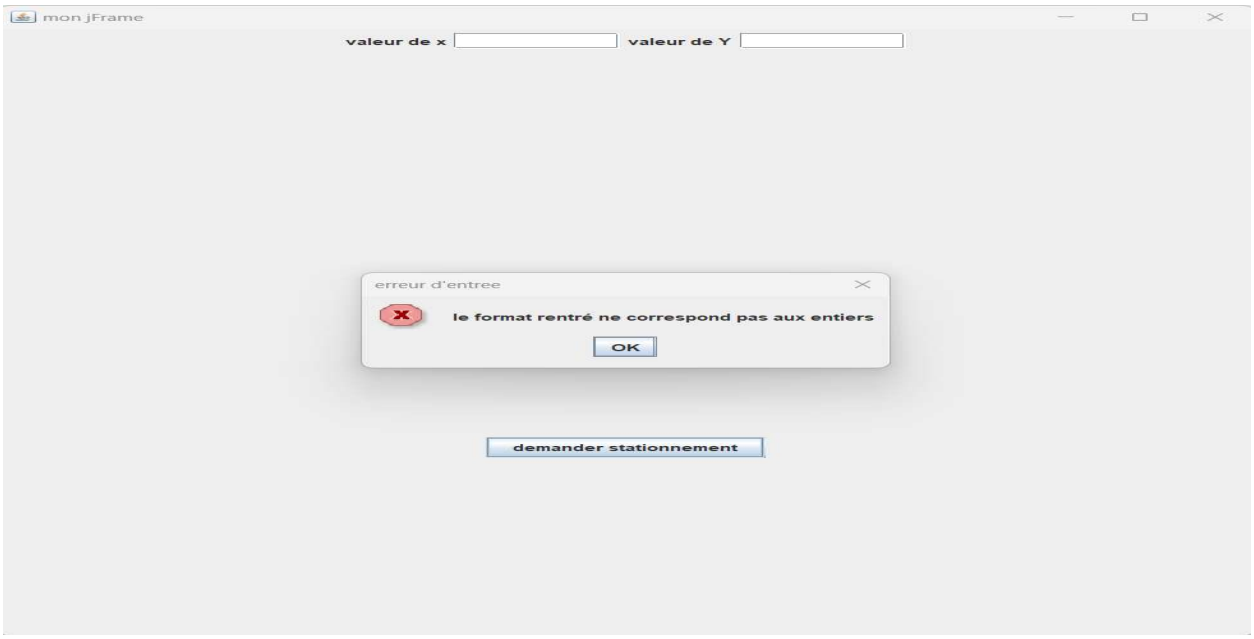
```java
    Graph graph=new Graph(edges,19);
    cout=findShortestPaths(graph,0, 19);// determination du plus court chemin
    pointtrajets();// receuillement des points du trajet
}

/**
 * cette methode permet de recuperer les stations disponibles
 */
public  void stationDispo() {
    stationsdisponibles=new ArrayList<Stationnement>();
    for(Stationnement s:stations) {
        if(s.getEtat()) {
            stationsdisponibles.add(s);
        }
    }
}
/**
 * cette methode prend en parametre une position et genere l'ID de la route sur laquelle se trouve le point
 * @param p
 * @return
 */
public int getcase(Position p) {
    int id=0;
    for(route s:routes) {
        if(s.appartient(p)) {//condition d'appartenance à la route
            id=s.getId();
            return id;
        }
    }
return id;
}
/**
 * cette methode enclenche l'ensemble du traitement menant à la determination du plus court trajet en prennant en entrée la position de la source
 * @param p
 */
public void traitement(Position p) {
    generate_graph(getcase( p));
}
```

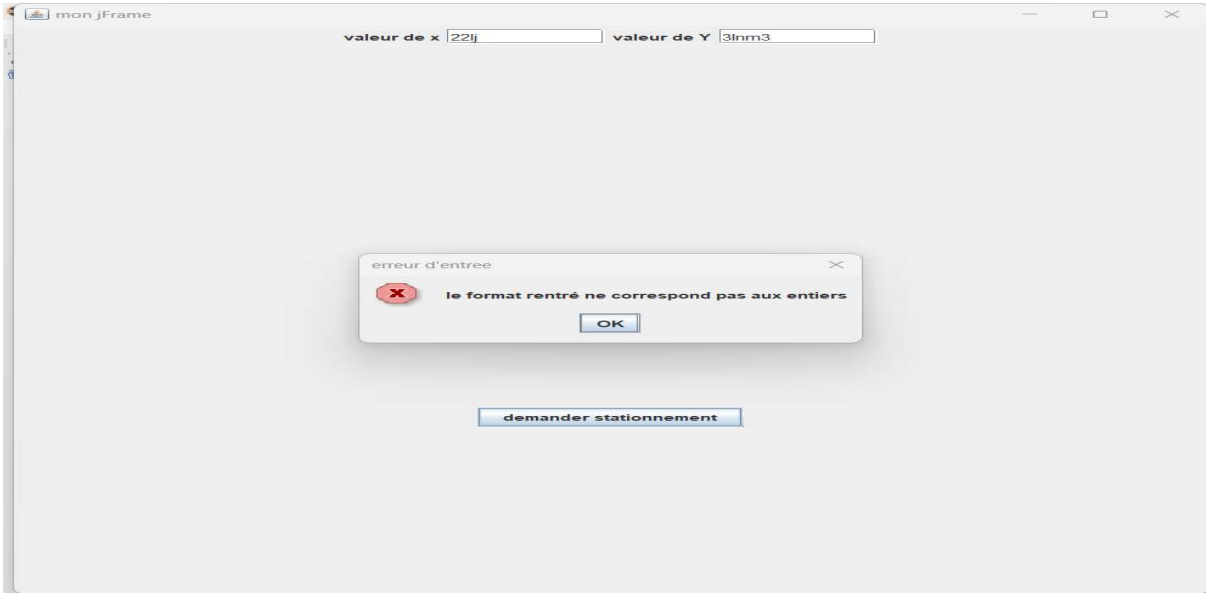
## 5. Tests de qualité

Rendu à la fin de notre développement, il est question de faire des tests de qualité.

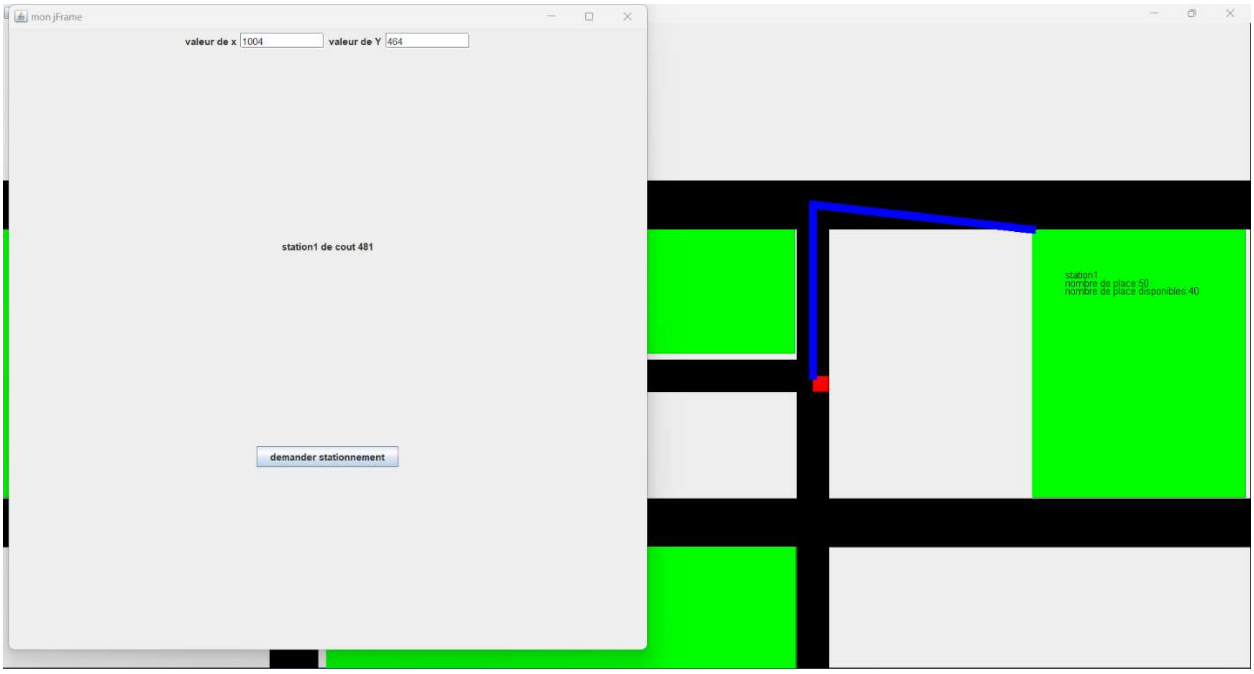
- Le premier test montre une exception car l'utilisateur n'as pas rentre de valeur dans le formulaire :




- Le deuxième test montre une seconde exception car l'user rentre des données de types différents des entiers.

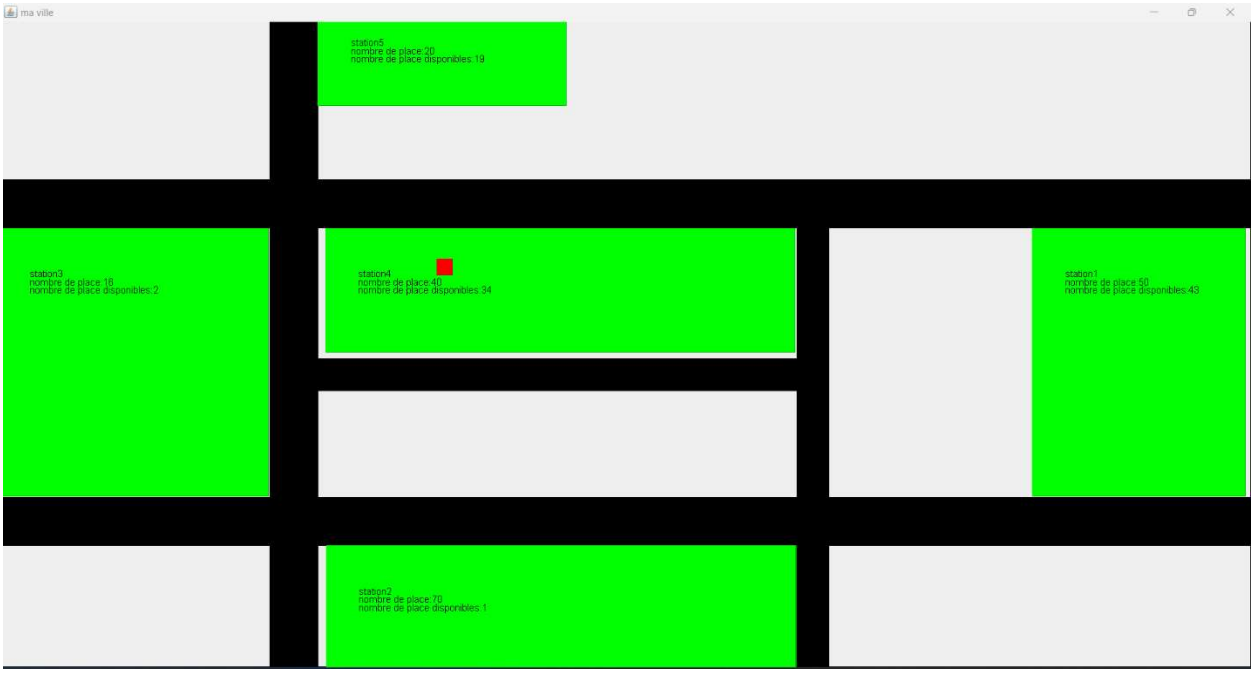


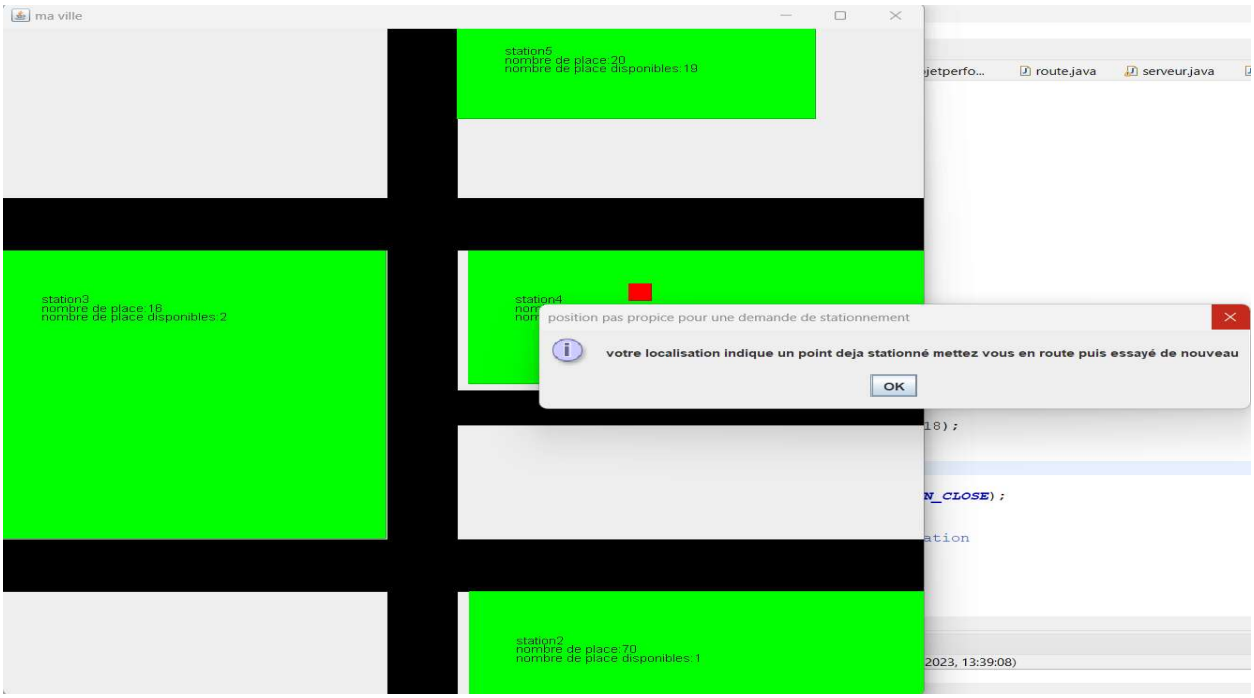
- Le troisième test illustre une réponse du serveur a l'utilisateur avec le nom de la station disponible et son cout le cout dans notre cas qui la distance uniquement et a cote nous pouvons voir le traçage vers la station 1




- La quatrième exception met en exergue le trace de la route vers la station4 dans un autre cas de figure.


- Le cinquième test présente un autre cas d'exception, on remarque que le conducteur est sur un point où il peut plus faire une demande de stationnement car il est déjà stationné.






Alors un message d'erreurs s'affiche à l'écran pour notifier l'utilisateur sur sa démarche pas logique.

# Conclusion

Pour terminer il était question pour nous dans ce projet de résoudre un problème de congestion lors du parking dans une ville. Ainsi, le travail réalisé portait sur l'analyse du problème, une conception, puis une implémentation suivie de tests de qualité. Travail étant effectué notre application pourra répondre à ce problème de congestion en basant la solution de l'application sur la distance uniquement. Après la réalisation du projet il est clair d'affirmer qu'il a été bénéfique pour nous car il a permis d'asseoir les notions d'association, encapsulation, qualités de programmations et programmation évènementielle. De plus la manipulation des différentes classes de java a permis une meilleure compréhension de ces dernières et ouvre un champ à de nouvelles perspectives de configuration d'interfaces et de gestion des évènements dans java.

## Références

Java Tutorial | Learn Core Java Programming Language. (s. d.). EDUCBA. https://www.educba.com/software-development/software-development-tutorials/java-tutorial/

Chemins les plus courts à source unique - Algorithme de Dijkstra. (s. d.). https://www.techiedelight.com/fr/single-source-shortest-paths-dijkstras-algorithm/

Farrell, J. (1999b). Java Programming : Comprehensive.