UNIVERSITÉ DE NANTES

Programmation Orientée Object
Rapport de projet
GANDIBLEUX Elizabeth et GUY-DEROUBAIX Valentin
381C/D
Licence de Mathématiques-Informatique
Décembre 2021

1 Introduction

Dans ce projet, nous avons réalisé un logiciel pour répondre aux besoins d'une petite entreprise de distribution de repas en milieu urbain. Dans un premier temps, elle souhaite acquérir des véhicules, d'abord des vélos et des scooters. Puis l'entreprise souhaite embaucher des salariés. Son activité principale c'est de réaliser des livraisons (c'est à dire des courses entreprise - clients) et sélectionner des courses les plus optimales. Pour visualiser tout cela, l'entreprise souhaite disposer d'une interface graphique. Le rapport présente (1) le diagrammes des classes, (2) la spécification des classes, (3) un exemple de résultat et (4) une conclusion.

2 Fonctionnement

L'application est divisée en une carte à gauche, des informations sur la première livraison en haut à droite et des boutons pour ajouter des employés et des véhicules. Pour initialiser une livraison il suffit de cliquer sur la carte à l'endroit voulu puis de sélectionner le poids de la livraison. L'animation se lancera aussitôt.

Le projet a été fait sous java 17.0.1 et javaFx 17.0.1.

3 Diagramme de classes

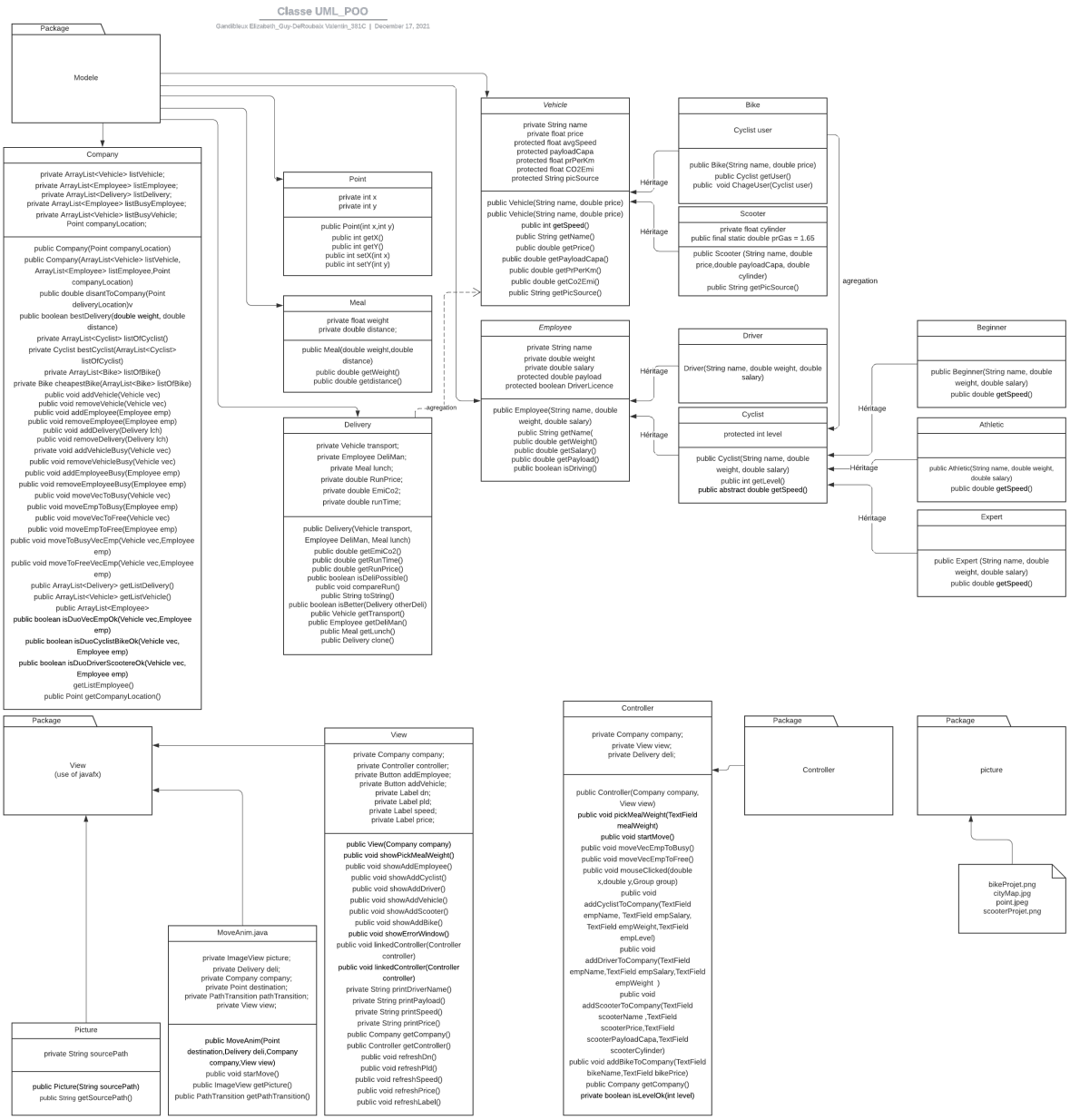

Figure 1 : Diagramme de classes du logiciel

4 Spécification

4.1 Package Modele

Classe : Vehicle
Rôle : La classe Vehicle est une classe abstraite. Elle est la classe mère de Scooter, et Bike.
Méthodes :

Signature	Description
public Vehicle(String name, double price,double payloadCapa)	Constructeur de la classe Vehicle il initialise le nom, le prix et la charge utile
public Vehicle(String name, double price)	Constructeur de la classe Vehicle il initialise le nom et le prix
public int getSpeed()	Getter de la vitesse du vehicule
public String getName()	Getter de nom du vehicule
public double getPrice()	Getter du prix du vehicule
public double getPayloadCapa()	Getter de la charge utile du vehicule
public double getPrPerKm()	Getter du prix par km
public double getCo2Emi()	Méthode qui retourne l'émission de CO2
public String getPicSource()	Getteur qui retourne la chemin de l'image associé au vehicule

Classe : Bike
Rôle : La classe Bike instancie Athletic, Beginner et Expert.
Méthodes :

Signature	Description
public Bike(String name, double price)	Constructeur
public Cyclist getUser()	Getter de user
public void changeUser(Cyclist user)	Méthode pour changer le user

Classe : Scooter
Rôle : La classe Scooter est une classe héritée de Vehicle.
Méthodes :

Signature	Description
public Scooter (String name, double price,double payloadCapa, double cylinder)	Constructeur de Scooter
public String getPicSource()	Getteur de picSource

Classe : Employee
Rôle : La classe Employee est une classe Abstraite. C'est la classe mère de cyclist et driver.
Méthodes :

Signature	Description
public Employee(String name, double weight, double salary)	Constructeur de Employee
public String getName()	Getter de nom de l'employé
public double getWeight()	Getter du poids de l'employé
public double getSalary()	Getter du salaire de l'employé
public double getPayload()	Getter de la charge utile de l'employé

Classe : Cyclist
Rôle : La classe Cyclist est une classe héritée de Employee.
Méthodes :

Signature	Description
public Cyclist(String name, double weight, double salary)	Constructeur de Cyclist
public int getLevel()	Getter qui retourne le niveau de l'employé cyliste
public double getSpeed()	Getter sur la vitesse des cyclists

Classe : Beginner
Rôle : La classe Beginner est une classe héritée de Cyclist.
Méthodes :

Signature	Description
public Beginner(String name, double weight, double salary)	Constructeur de Beginner

Classe : Athletic
Rôle : La classe Athletic est une classe héritée de Cyclist.
Méthodes :

Signature	Description
public Athletic(String name, double weight, double salary)	Constructeur de Athletic

Classe : Expert
Rôle : La classe Expert est une classe héritée de Bike.
Méthodes :

Signature	Description
public Expert (String name, double weight, double salary)	Constructeur de Expert

Classe : Driver
Rôle : La classe Driver est une classe héritée de Employee.
Méthodes :

Signature	Description
Driver(String name, double weight, double salary)	Constructeur de Diver

Classe : Company

Rôle : La classe Company est une classe qui répertorie tous les véhicules et salariés de l'entreprise.

Méthodes :

Signature	Description
public Company(Point companyLocation)	Constructeur qui initialise toutes les listes à vide
public Company(ArrayList<Vehicle> listVehicle, ArrayList<Employee> listEmployee,Point companyLocation)	Constructeur qui initialise les listes de salariés et de véhicules avec les listes passées en paramètres
public double disantToCompany(Point deliveryLocation)	Méthode qui calcule la distance à parcourir
public boolean bestDelivery(double weight, double distance)	Méthode qui détermine qu'elle est la course la plus optimale
private ArrayList<Cyclist> listOfCyclist()	Méthode qui répertorie la liste de cyclistes non disponibles
public void addVehicle(Vehicle vec)	Méthode qui ajoute un véhicule dans la liste des véhicules disponibles pour effectuer une course
public void removeVehicle(Vehicle vec)	Méthode qui retire un véhicule dans la liste des véhicules disponibles pour effectuer une course
public void addEmployee(Employee emp)	Méthode qui ajoute un employé dans la liste des employés disponibles pour effectuer une course
public void removeEmployee(Employee emp)	Méthode qui retire un employé dans la liste des employés disponibles pour effectuer une course
public void addDelivery(Delivery lch)	Méthode qui ajoute une livraison dans la liste des livraisons disponibles
public void removeDelivery(Delivery lch)	Méthode qui retire une livraison dans la liste des livraisons disponibles
private void addVehicleBusy(Vehicle vec)	Méthode qui ajoute un véhicule dans la liste des véhicules non disponibles entrain d'effectuer une livraison
public void removeVehicleBusy(Vehicle vec)	Méthode qui retire un véhicule dans la liste des véhicules non disponibles car ils ont fini d'effectuer une livraison
public void addEmployeeBusy(Employee emp)	Méthode qui ajoute un employé dans la liste des employés non disponibles pour effectuer une course

public void removeEmployeeBusy(Employee emp)	Méthode qui retire un employé dans la liste des employés non disponibles car il l'est de nouveau
public void moveVecToBusy(Vehicle vec)	Méthode qui déplace un véhicule libre vers la liste des véhicules non disponibles
public void moveEmpToBusy(Employee emp)	Méthode qui déplace un employé libre vers liste des employés non disponibles
public void moveVecToFree(Vehicle vec)	Méthode qui déplace un véhicule non disponible vers la liste des véhicules libres
public void moveEmpToFree(Employee emp)	Méthode qui déplace un employé non disponible vers la liste des employés libres
public void moveToBusyVecEmp(Vehicle vec,Employee emp)	Méthode qui déplace un véhicule et un employé libre vers la liste des véhicules non disponibles et des employés non disponibles
public void moveToFreeVecEmp(Vehicle vec,Employee emp)	Méthode qui déplace un véhicule et un employé occupé vers la liste des véhicules disponibles et des employés disponibles
public ArrayList<Delivery> getListDelivery()	Méthode qui retourne la liste des livraisons
public ArrayList<Vehicle> getListVehicle()	Méthode qui retourne la liste des véhicules
public ArrayList<Employee> getListEmployee()	Méthode qui retourne la liste des employés
public boolean isDuoVecEmpOk(Vehicle vec,Employee emp)	Méthode qui vérifie si l'employé peut utiliser le véhicule
public boolean isDuoCyclistBikeOk(Vehicle vec, Employee emp)	Méthode qui vérifie si l'employé est un cycliste et le véhicule un vélo
public boolean isDuoDriverScootereOk(Vehicle vec, Employee emp)	Méthode qui vérifie si l'employé est un conducteur et le véhicule un scooter
public Point getCompanyLocation()	Méthode qui retourne les coordonées de l'entreprise

Classe : Point
Rôle : La classe Point sert à modéliser les coordonées gps de différents points.
Méthodes :

Signature	Description
public Point(int x,int y)	Constructeur de Point
public int getX()	Getter de x
public int getY()	Getter de y
public int setX(int x)	Setter de x
public int setY(int y)	Setter de y

Classe : Meal
Rôle : La classe Meal sert à modéliser le repas.
Méthodes :

Signature	Description
public Meal(double weight,double distance)	Constructeur de Meal
public double getWeight()	Getter du poids du repas
public double getdistance()	Getter de la distance à parcourir

Classe : Delivery
Rôle : La classe Delivery est une classe .
Méthodes :

Signature	Description
public Delivery(Vehicle transport, Employee DeliMan, Meal lunch)	Constructeur de Delivery
public boolean isDeliPossible()	Getter du poids du repas
public boolean isBetter(Delivery otherDeli)	Getter de la distance à parcourir
public Vehicle getTransport()	Getter du véhicule utilisé
public Employee getDeliMan()	Getter du salarié utilisé
public Meal getLunch()	Getter du repas transporté
public double getRunPrice()	Getter du prix de la course
public double getEmiCo2()	Getter de l'émission de co2 utilisé par le transporteur
public double getRunTime()	Getter du prix de l'émission de la course
public Delivery clone()	Méthode qui retourne un clone de la livraison
public String toString()	Méthode qui affiche toutes les informations du livreur et la course

4.2 Package View

Classe : picture
Rôle : Le package picture est une classe .
Méthodes :

Signature	Description
public Picture(String sourcePath)	Constructeur de picture
public String getSourcePath()	Méthode qui retourne la source de la photo utilisée

Classe : MoveAnim
Rôle : Le Classe picture est une classe .
Méthodes :

Signature	Description
public MoveAnim(Point destination,Delivery deli,Company company,View view)	Constructeur de MoveAnim, le constructeur initialise aussi l'animation
public void starMove()	Méthode qui permet de commencer l'animation
public ImageView getPicture()	Méthode qui retourne la photo utilisée
public PathTransition getPathTransition()	Getteur de l'attribut PathTransition

Classe : View
Rôle : Le Classe View est une classe .
Méthodes :

Signature	Description
public View(Company company)	Constructeur de la view
public void showPickMealWeight()	Méthode qui permet via l'interface graphique de sélectionner le poids du repas
public void showAddEmployee()	Méthode qui permet via l'interface graphique d' ajouter un employé
public void showAddCyclist()	Méthode qui permet via l'interface graphique d' ajouter un cycliste
public void showAddDriver()	Méthode qui permet via l'interface graphique d' ajouter un conducteur
public void showAddVehicle()	Méthode qui permet via l'interface graphique d' ajouter un véhicule
public void showAddScooter()	Méthode qui permet via l'interface graphique d' ajouter un scooter
public void showAddBike()	Méthode qui permet via l'interface graphique d' ajouter un vélo
public void showErrorWindow()	Méthode qui permet de gérer les exceptions d'information mal rentrées
private String printDriverName()	Méthode qui affiche sur la console le nom de l'employé de la première livraison dans la liste
private String printPayload()	Méthode qui affiche sur la console la charge utile de l'employé de la première livraison dans la liste
private String printSpeed()	Méthode qui affiche sur la console la vitesse de l'employé de la première livraison dans la liste
private String printPrice()	Méthode qui affiche sur la console le prix de la course de l'employé de la première livraison dans la liste
private String printCo2()	Méthode qui affiche sur la console le taux de co2 libéré de la course par l'employé de la première livraison dans la liste
public Company getCompany()	Getteur de company
public Controller getController()	Getteur du controlleur
public void refreshDn()	Méthode qui lors du rafraîchissement, rafraîchit et affiche sur l'interface graphique le nom du conducteur
public void refreshPld()	Méthode qui lors du rafraîchissement, rafraîchit et affiche sur l'interface graphique la charge utilise du salarié
public void refreshSpeed()	Méthode qui lors du rafraîchissement, rafraîchit et affiche sur l'interface graphique la vitesse du salarié
public void refreshPrice()	Méthode qui lors du rafraîchissement, rafraîchit et affiche sur l'interface graphique le prix de la course du salarié
public void refreshCo2()	Méthode qui lors du rafraîchissement, rafraîchit et affiche sur l'interface graphique le taux de co2 libéré par le salarié
public void refreshLabel()	Méthode qui fait appel aux méthodes de rafraichissement des attributs

4.3 Package Controller

Classe : Controller
Rôle : La classe Controller est une classe .
Méthodes :

Signature	Description
public Controller(Company company, View view)	Constructeur du Controller
public void moveVecEmpToBusy()	Méthode qui déplace l'employé durant sa course sur l'interface graphique
public void pickMealWeight(TextField mealWeight)	Méthode qui assigne à l'attribut weight du controller la valeur saisie par l'utilisateur
public void startMove()	Méthode qui déclenche l'animation du véhicule sur la carte
public void mouseClicked(double x,double y,Group group)	Méthode qui appele toutes les fonctions nécessaire au lancement de l'animation
public void addCyclistToCompany(TextField empName, TextField empSalary, TextField empWeight,TextField empLevel)	Méthode qui ajoute un cycliste via la saisie de l'interface graphique
public void addDriverToCompany(TextField empName,TextField empSalary,TextField empWeight)	Méthode qui ajoute une livraison via la saisie de l'interface graphique
public void addScooterToCompany(TextField scooterName ,TextField scooterPrice,TextField scooterPayloadCapa,TextField scooterCylinder)	Méthode qui ajoute un scooter via la saisie de l'interface graphique
public void addBikeToCompany(TextField bikeName,TextField bikePrice)	Méthode qui ajoute un vélo via la saisie de l'interface graphique
public Company getCompany()	Getteur de l'entreprise
private boolean isLevelOk(int level)	Méthode qui vérifie si le niveau d'un cycliste est bien entre 1 et 3 compris

4.4 Package picture

Package : picture
Rôle : Le package picture est une classe .
Méthodes :

Signature	Description
bikeProjet.png.	image du vélo
cityMap.jpg.	image d'arrière fond
point.jpeg.	image du point
scooterProjet.png	image du scooter

5 Résultats

5.1 Jeux de tests

Exemple Modele :

distance en km	poids du repas en kg	prix de la course en €
type de véhicule	nom du salarié	vitesse du salarié en km/h

Résultats aux jeux de tests :

13,5 km	23 kg	49€
scooter	Tiago	36km/h

31,6 km	90 kg	116€
scooter	Tiago	36km/h

18,5 km	20 kg	67€
scooter	Marcel	36km/h

Nous pouvons remarquer que l'application enverra toujours en premier des scooters jusqu'à ce qu'il ne reste plus que des vélos.

6 Conclusion

Pour conclure, notre logiciel permet de calculer la course la plus optimale et de fournir une interface graphique à l'entreprise. Il est un extensible puisque l'ajout de salariés, de véhicules et autres ne nécessite pas de modification majeure du code. Cependant dans notre code, notre fonction qui calcule la course la plus optimale fait des vérifications de classes. Donc ajouter des classes qui héritent de Vehicle ou Employee ajouterait des conditions qu'il faudrait implémenter. Il y aura autant de conditions que de combinaisons de véhicules/employés pouvant partir en livraison, ce qui n'est pas extensible si l'on veut ajouter plusieurs classes différentes. Certains bugs de démarage on été remarqués ; il s'agit généralement de la bibliblothèque javafx17.1 qui est une bibliothèque utilisateur qui doit être changée par la bibliothèque utilisateur javafx de l'ordinateur. Le principe d'encapsulation est respecté dans le code. Il est possible de lancer plusieurs animations en même temps. Pour les améliorations possibles nous pourrions proposer à l'utilisateur un menu où il sélectionnerait la ville et l'employé de son choix (si il y a du choix). On pourrait aussi ajouter un endroit sur l'interface graphique où l'utilisateur pourrait donner un pourboire.