1) But

Le but de ce travail est de mettre en pratique les concepts de la programmation Java à travers l'implantation des règles d'affaires (validation, calcul, etc.) correspondant aux spécifications du client. En outre, l'étudiant aura à définir ses propres classes et leurs méthodes tout en respectant les conventions d'écriture du code Java vues en classe.

2) Description des besoins du client

Le respect de l'échéancier et la qualité des premières versions livrées ont encouragé Simba le propriétaire de de l'entreprise **Roulons les Véhicules Verts** (RVV) à vous confier la suite du développement de son logiciel de facturation. Cette nouvelle version du logiciel réutilisera presque toutes les méthodes de saisie et de validations implantées dans le travail pratique 2 et bénéficiera du paradigme orienté-objet très avancé avec l'implantation de plusieurs classes dont les détails sont donnés dans les sections suivantes.

3) Les spécifications

3.1. Fonctionnement du menu d'options

Au démarrage, le programme doit lire le fichier `InventaireVehicules.csv` qui contient les informations sur les types et les grandeurs des véhicules, les prix de location et d'assurance des véhicules par jour, le nombre de véhicules disponibles par type et par grandeur. Ensuite, il doit afficher un message de bienvenue et un menu des 5 options suivantes :

1. Facturer la location des véhicules
2. Afficher l'inventaire des véhicules
3. Afficher le nombre de véhicules hybrides et électriques loués
4. Afficher toutes les factures
5. Quitter le programme

Les descriptions de ces options sont :

Option 1 : Elle permet la facturation de la location de plusieurs véhicules.

1. Pour chaque type et grandeur de véhicule à louer, votre programme doit saisir :
    a. Le type du véhicule à louer (H ou h pour Hybride, et E ou e pour Électrique).
    b. La grandeur du véhicule à louer (P ou p pour Petit, I ou i pour Intermédiaire, et G ou g pour Grand).
    c. Le nombre de véhicules à louer (entre 0 et 5 inclusivement).
        - ✓ Si le nombre de véhicules à louer est inférieur à 0 ou supérieur à 5, un message d'erreur est affiché, et votre programme doit demander à l'utilisateur d'entrer de nouveau un nombre entre 0 et 5 inclusivement.
        - ✓ Si le nombre de véhicules à louer est égal à 0, la location de ce type et de cette grandeur de véhicule est annulée, et un message d'information est affiché. S'il n'y a aucun autre type et aucune autre grandeur de véhicule loué, votre programme doit demander à l'utilisateur d'appuyer sur <ENTRÉE> pour réafficher le menu principal.
        - ✓ Si le nombre de véhicules est entre 1 et 5 inclusivement, votre programme doit vérifier si le nombre de véhicules à louer est inférieur ou égal au nombre de véhicules disponibles dans l'inventaire (le même type, et la même grandeur), sinon un message d'erreur doit être affiché, et votre programme doit demander à l'utilisateur d'entrer un nouveau nombre de véhicules à louer.

   d. Le nombre de jours de location (supérieur à 0 et inférieur ou égal à 30).

   e. Votre programme doit demander si le locataire désire prendre une assurance pour cette location. Les réponses acceptées sont O ou o pour Oui, N ou n pour Non.

2. Votre programme doit demander si le locataire désire louer un autre type et une autre grandeur de véhicule à louer. Les réponses acceptées sont O ou o pour Oui, N ou n pour Non. Si la réponse est Oui, votre programme doit répéter les étapes a, b, c, d, et e du point 1.

3. Si le nombre de véhicules à louer est supérieur à 1, votre programme doit saisir respectivement le prénom du locataire, le nom du locataire, le numéro de téléphone du locataire, le numéro de permis de conduire du locataire, le mode de paiement (D ou d pour Débit, C ou c pour Crédit).

   Si le mode de paiement est crédit, votre programme doit aussi saisir le type de la carte de crédit (V ou v pour Visa, et M ou m pour MasterCard), et le numéro de la carte de crédit.

4. Votre programme doit calculer la facture de la location des véhicules choisis. Les règles de calcul sont les mêmes que celles du TP2.

5. Votre programme doit afficher le nom de l'entreprise, le numéro de téléphone de l'entreprise ((438) 222-1111), l'adresse de l'entreprise ((1500 rue Hakuna, Matata, Québec Y0Z 6Y7)), le numéro de la facture, la date et l'heure courantes, les informations du locataire saisies, et les détails de la facture. Pour plus de détails sur l'affichage, voir les exemples de la trace d'exécution du programme fournis avec l'énoncé du Travail pratique 3.

   Après l'affichage de la facture, votre programme doit ajouter la facture dans la liste des factures. Enfin, votre programme doit demander à l'utilisateur d'appuyer sur <ENTRÉE> pour réafficher le menu principal.

Option 2 :

Votre programme doit afficher les nombres de véhicules disponibles par type et par grandeur de véhicule lus dans le fichier `InventaireVehicules.csv`. Ces nombres changeront au fur et à mesure que les véhicules seront loués.

Option 3 :

Votre programme doit afficher le nombre de véhicules hybrides et électriques loués par type et par grandeur. Pour plus de détails sur l'affichage, voir les exemples de la trace d'exécution du programme fournis avec l'énoncé du Travail pratique 3. Ensuite, votre programme doit demander à l'utilisateur d'appuyer sur <ENTRÉE> pour réafficher le menu principal.

Option 4 :

Votre programme doit afficher toutes les factures créées. Si le nombre de factures à afficher est supérieur à 1, votre programme doit demander à l'utilisateur d'appuyer sur <ENTRÉE> pour continuer l'affichage des factures.

Après l'affichage de toutes les factures, votre programme doit demander à l'utilisateur d'appuyer sur <ENTRÉE> pour réafficher le menu principal. Pour plus de détails sur l'affichage, voir les exemples de la trace d'exécution du programme fournis avec l'énoncé du Travail pratique 3.

Option 5 :

Toutes les données de toutes les factures générées doivent être écrites dans le fichier `Factures.csv`. Votre programme doit afficher le message de remerciement et prendre fin.

3.2. Les classes à implémenter

Nom de la classe : `Locataire`		
Attributs ou variables (`private`**)**		
Nom	**Type**	**Description**
Le nom	`String`	Le nom du locataire.
Le prénom	`String`	Le prénom du locataire.
Le numéro de téléphone	`String`	Le numéro de téléphone du locataire.
Le numéro de permis de conduire	`String`	Le numéro de permis de conduire du locataire.
Constructeurs et les méthodes d'objet ou d'instance (toutes ces méthodes sont « `public` **» et « non** `static` **»)**		
Un constructeur avec les paramètres suivants : le nom, le prénom, le numéro de téléphone, et le numéro de permis de conduire.		
Les méthodes (getters) pour retourner les attributs : le nom, le prénom, le numéro de téléphone, et le numéro de permis de conduire. Ces méthodes ne reçoivent aucun paramètre.		

Nom de la classe : `Vehicule`		
Les constantes (`« public », « static »` **et** `« final »`**)**		
Le type et la grandeur des véhicules (H, E, P, I, et G) de type `char`, et les descriptions du type et de la grandeur des véhicules (Hybride, Électrique, Petit, Intermédiaire, et Grand) de type `String`.		
Attributs ou variables (`private`**)**		
Nom	**Type**	**Description**
Le type	`char`	Le type du véhicule.
La grandeur	`char`	La grandeur du véhicule
Le prix de la location par jour	`float`	Le prix de la location par jour de ce type et de cette grandeur.
Le prix de l'assurance par jour	`float`	Le prix de l'assurance par jour de ce type et de cette grandeur.
Constructeurs et les méthodes d'objet ou d'instance (toutes ces méthodes sont « `public` **» et « non** `static` **»)**		
Un constructeur avec les paramètres suivants : le type, la grandeur, les prix de la location et de l'assurance par jour.		
Les méthodes (getters) pour retourner les attributs : le type, la grandeur, le prix de la location par jour, et le prix de l'assurance par jour. Ces méthodes ne reçoivent aucun paramètre.		
Une méthode pour obtenir la description du type de véhicule. Elle ne reçoit aucun paramètre, et elle doit retourner la description du type de véhicule. Exemple : si le type du véhicule = `'H'`, la méthode doit retourner = `"Hybride"`.		
Une méthode pour obtenir la description de la grandeur du véhicule. Elle ne reçoit aucun paramètre, et elle doit retourner la description de la grandeur du véhicule. Exemple : si la grandeur du véhicule = `'P'`, la méthode retourne = `"Petit"`.		

Nom de la classe : `VehiculeLoue`		
Les constantes (`« public », « static »` **et** `« final »`**)**		
Le taux de rabais (20%) de type `float` et le nombre de jours requis pour appliquer le rabais (15) de type `byte`.		
Attributs ou variables (`private`**)**		
Nom	**Type**	**Description**
Le véhicule	`Vehicule`	Le véhicule loué
Le nombre de véhicules loués	`int`	Le nombre de véhicules loués de ce type et de cette grandeur.
Le nombre de jours de location	`int`	Le nombre de jours de location du véhicule
La date de la location	`LocalDateTime`	La date de location du véhicule
Constructeurs et les méthodes d'objet ou d'instance (toutes ces méthodes sont « `public` **» et « non** `static` **»)**		
Un constructeur avec les paramètres suivants : le véhicule, la date de location, le nombre de véhicules loués, et le nombre de jours de location.		
La méthode (setter) pour modifier le nombre de véhicules loués. Cette méthode reçoit en paramètre la nouvelle valeur du nombre de véhicules loués et elle ne retourne aucune valeur.		
Les méthodes (getters) pour retourner les attributs : le véhicule, le nombre de véhicules loués, le nombre de jours de location, et la date de location. Ces méthodes ne reçoivent aucun paramètre.		
Une méthode pour calculer la date de retour du véhicule. Elle ne reçoit aucun paramètre, et elle doit retourner la date de retour. La date de retour est calculée en ajoutant le nombre de jours de location à la date de la location du véhicule.		
Une méthode pour calculer le rabais. Elle ne reçoit aucun paramètre, et elle doit retourner le rabais calculé. Si le nombre de jours de location est supérieur à 15 jours, le type de véhicule choisi est électrique, et la grandeur du véhicule choisi est petit ou intermédiaire, la méthode doit calculer un rabais de 20% sur le prix de la location du véhicule par jour.		

Nom de la classe : `VehiculeDisponible`		
Attributs ou variables (`private`**)**		
Nom	**Type**	**Description**
Le véhicule	`Vehicule`	Le véhicule disponible
Le nombre de véhicules disponibles	`int`	Le nombre de véhicules disponibles.
Constructeurs et les méthodes d'objet ou d'instance (toutes ces méthodes sont « `public` **» et « non** `static` **»)**		
Un constructeur avec les paramètres suivants : le véhicule, et le nombre de véhicules disponibles.		
Une méthode (setter) pour modifier le nombre de véhicules disponibles. Cette méthode reçoit en paramètre la nouvelle valeur de l'attribut à modifier.		
Les méthodes (getters) pour retourner les attributs : le véhicule, et le nombre de véhicules disponibles. Ces méthodes ne reçoivent aucun paramètre.		

Nom de la classe : `LocationVehicule`		
Les constantes (« `public` », « `static` » **et** « `final` »**)**		
Le nombre maximum de véhicules à louer de type `int`, dont la valeur est 50.		
Attributs ou variables (`private`**)**		
Nom	**Type**	
Le locataire	`Locataire`	Le locataire du véhicule.
Le tableau des véhicules loués	`[] VehiculeLoue`	Le tableau des véhicules loués par ce locataire.
Constructeurs et les méthodes d'objet ou d'instance (toutes ces méthodes sont « `public` **» et « non** `static` **»)**		
Un constructeur sans paramètre. La création du tableau 'le tableau des véhicules loués' doit se faire à l'intérieur du constructeur et la taille de ce tableau doit être égale au nombre maximum de véhicules à louer.		
Les méthodes (getters) pour retourner les attributs : le locataire, et le tableau des véhicules loués. Ces méthodes ne reçoivent aucun paramètre.		
Une méthode pour obtenir le nombre de types de véhicules loués. Elle ne reçoit aucun paramètre, elle doit compter le nombre de types de véhicules loués dans le tableau des véhicules loués, et retourner ce nombre. Ce nombre servira à connaitre le nombre d'éléments dans le tableau des véhicules loués et il servira aussi d'indice du tableau pour le prochain ajout (la position du tableau des véhicules loués à laquelle il faut insérer le prochain véhicule loué).		
Une méthode pour ajouter un véhicule loué dans le tableau des véhicules loués. Elle reçoit en paramètre un véhicule loué, et elle ajoute ce véhicule dans la prochaine position libre du tableau des véhicules loués. Cette position libre doit être inférieure à la taille du tableau des véhicules loués. La méthode doit retourner vrai si le véhicule est ajouté, sinon faux.		
Une méthode pour obtenir la position d'un véhicule loué dans 'le tableau des véhicules loués'. Elle reçoit en paramètres le type et la grandeur du véhicule loué. Elle parcourt 'le tableau des véhicules loués' pour trouver la position du véhicule loué dont le type est égal au type passé en paramètre, et la grandeur est égale à la grandeur passée en paramètre. Elle doit retourner la position du véhicule, ou -1 si elle ne trouve aucun véhicule loué de ce type et de cette grandeur.		

Nom de la classe : `Facture`		
Les constantes (« `public` », « `static` » **et** « `final` »**)**		
Les modes de paiement (D, et C) de type `char`, et les descriptions des modes de paiement (Débit, et Crédit) de type `String`. Les types de carte de crédit (V, et M) de type `char`, et les descriptions des types de carte de crédit (Visa, et MasterCard) de type `String`. Les taux TPS et TVQ de type `float`. Les informations de l'entreprise (le nom, l'adresse, et le téléphone) de type `String`. Le format de la date de type `DateTimeFormatter`.		
Attributs ou variables (`private`**)**		
Nom	**Type**	**Description**
Le compteur du nombre de factures	`int`	Un nombre entier qui doit être incrémenté à chaque création d'une nouvelle facture. Cet attribut doit être « `static` », et il doit être initialisé à 0 dès sa déclaration.
Le numéro de la facture	`int`	Un nombre entier qui représente le numéro de la facture.
Le mode de paiement	`char`	Le mode de paiement choisi de cette facture.
Le type de la carte de crédit	`Char`	Le type de la carte de crédit si le paiement est fait par carte de crédit.
Le numéro de la carte de crédit	`String`	Le numéro de la carte de crédit si le paiement est fait par carte de crédit.
La date de la facture	`LocalDateTime`	La date de la facture courante
La location de véhicule	`LocationVehicule`	Elle représente toutes les informations du locataire et des véhicules qu'il a loués.
Le sous-total	`float`	Un nombre réel qui représente le sous-total de la facture.
Le montant TPS	`float`	Un nombre réel qui représente le montant de la taxe TPS.

Le montant TVQ	`float`	Un nombre réel qui représente le montant de la taxe TVQ.
Le montant total	`float`	Un nombre réel qui représente le montant total de la facture.
Constructeurs et les méthodes d'objet ou d'instance (toutes ces méthodes sont « `public` » et « non `static` »)		
Un constructeur avec les attributs : la date de la facture, la location de véhicule et le mode de paiement. L'incrémentation de l'attribut 'le compteur du nombre de la facture' doit être faite à l'intérieur du constructeur, et la nouvelle valeur de ce compteur après l'incrémentation doit être affectée à l'attribut 'le numéro de la facture'.		
Les méthodes (setters) pour modifier le type de la carte de crédit, et le numéro de la carte de crédit. Ces méthodes reçoivent en paramètre la nouvelle valeur de l'attribut à modifier.		
Les méthodes (getters) pour retourner les attributs : la date de la facture, la location de véhicule, le mode de paiement, le type de la carte de crédit, le numéro de la carte de crédit, le sous-total, le montant de la taxe TPS, le montant de la taxe TVQ, le sous-total, le montant total. Ces méthodes ne reçoivent aucun paramètre.		
Une méthode pour obtenir la description du mode de paiement. Elle ne reçoit aucun paramètre, et elle doit retourner la description du mode de paiement. Exemple : si le mode de paiement = `'D'`, la méthode retourne = `"Débit"`.		
Une méthode pour obtenir la description du type de la carte de crédit. Elle ne reçoit aucun paramètre, et elle doit retourner la description du type de la carte de crédit. Exemple : si le type de la carte de crédit est = `'V'`, la méthode retourne = `"Visa"`.		
Une méthode pour calculer le sous-total de la facture. Cette méthode doit parcourir le tableau des véhicules loués de l'attribut 'la location de véhicule'. Pour chaque véhicule loué : 1) Elle doit multiplier le prix de la location par jour moins le rabais par le nombre de jours de location et le nombre de véhicules loués. 2) Elle doit multiplier le prix de l'assurance par jour par le nombre de jours de location et le nombre de véhicules loués. 3) Elle doit additionner les montants obtenus aux points 1) et 2). Elle doit ensuite faire la somme de tous les montants obtenus par véhicule loué. Cette somme doit être affectée à l'attribut 'le sous-total'. Cette méthode ne prend aucun paramètre, et ne retourne aucune valeur.		
Une méthode pour calculer le montant TPS. Cette méthode se base sur le sous-total de la facture et le taux de la taxe TPS (5%) pour calculer le montant de la taxe et le mettre dans l'attribut 'le montant TPS'. Cette méthode ne prend aucun paramètre, et ne retourne aucune valeur.		
Une méthode pour calculer le montant TVQ. Cette méthode se base sur le sous-total de la facture et le taux de la taxe TVQ (9.975%) pour calculer le montant de la taxe et le mettre dans l'attribut 'le montant TVQ'. Cette méthode ne prend aucun paramètre, et ne retourne aucune valeur.		
Une méthode pour calculer le montant total. Cette méthode fait la somme du montant TPS, du montant TVQ, du sous-total. Cette nouvelle valeur est stockée dans l'attribut 'le montant total'. Cette méthode ne prend aucun paramètre, et ne retourne aucune valeur.		
Une méthode pour afficher exactement la facture comme montrée dans la trace d'exécution. Cette méthode ne prend aucun paramètre, et ne retourne aucune valeur. Pour plus de détails sur l'affichage, voir les exemples de la trace d'exécution du programme fournis avec l'énoncé du Travail pratique 3.		

3.3. Les classes à compléter

Le nom de la classe : `GestionVehiculesDisponibles`
La description : Cette classe gère la liste des véhicules disponibles dans l'inventaire pour la location. Elle contient le nom du fichier à lire (`InventaireVehicules.csv`), et le tableau des véhicules disponibles. Dans le fichier à lire, excepté la première ligne qui est la description des données des autres lignes, chaque ligne est composée du type, de la grandeur, du prix de la location par jour, du prix de l'assurance par jour, et du nombre de véhicules disponibles. Chacune de ces lignes doit être lue et découpée pour créer un objet de type `VehiculeDisponible`, et cet objet doit être ajouté dans le tableau des véhicules disponibles. La première ligne doit être ignorée lors de la lecture. Voir le fichier `InventaireVehicules.csv` pour plus détails.
Les méthodes à compléter
✓ `lireFichierVehiculesDisponibles` ✓ `obtenirPrixLocationVehParJour` ✓ `obtenirPrixAssuranceVehParJour` ✓ `diminuerNbVehiculesDisponibles` ✓ `obtenirNbVehiculesDisponibles` ✓ `estDisponible` ✓ `afficher`
Les règles d'implémentation
Veuillez consulter les commentaires d'entête de ces méthodes (Javadoc) pour avoir plus de détails sur l'implémentation.

Le nom de la classe : `StatistiquesVehiculesLoues`
La description : Cette classe gère les données sur les nombres de véhicules loués par type de véhicule. Elle contient deux attributs `static` qui sont respectivement le nombre de véhicules hybrides loués et le nombre de véhicules électriques loués.
Les méthodes à compléter
✓ `augmenterNbVehiculesLoues` ✓ `obtenirNbVehiculesLoues` ✓ `afficherNbVehiculesLoues`
Les règles d'implémentation
Veuillez consulter les commentaires d'entête de ces méthodes (Javadoc) pour avoir plus de détails sur l'implémentation.

Le nom de la classe : `ListeDesFactures`
La description : Cette classe gère la liste de toutes les factures générées lors de la location des véhicules. Elle contient le nom du fichier dans lequel les factures seront sauvegardées (`Factures.csv`), le tableau des factures, le nombre courant de factures dans le tableau des factures, le nombre maximum de factures, et la description des données à écrire dans le fichier (entête). À la fin du programme, toutes les données des factures seront écrites dans le fichier `Factures.csv`.
Les méthodes à compléter
✓ `ajouterFacture` ✓ `ecrireFacture` ✓ `afficher`
Les règles d'implémentation
Veuillez consulter les commentaires d'entête de ces méthodes (Javadoc) pour avoir plus de détails sur l'implémentation.

Nom de la classe : `ApplicationPrincipale`
Méthodes (toutes ces méthodes sont "`public`" et "`static`")
La méthode `main` et toutes les méthodes d'affichage de menus, de saisies et de validations définies dans le cadre du travail pratique 2 doivent être dans cette classe. Ces méthodes sont : l'affichage du message de bienvenue, la saisie et validation de l'option choisie par l'utilisateur, la saisie et validation du prénom du locataire, la saisie et validation du nom du locataire, la saisie et validation du numéro de téléphone du locataire, la saisie et validation du numéro de permis de conduire, la saisie et validation du type de véhicule, la saisie et validation de la grandeur du véhicule, la saisie et validation du nombre de jours de location, la saisie et validation du mode de paiement, la saisie et validation du type de la carte de crédit, la saisie et validation du numéro de la carte de crédit, la saisie et validation de la réponse de la question si le locataire désire prendre un assurance.

4. Travail demandé

4.1. Organisation de votre programme

1. Dans la classe `ApplicationPrincipale` :

   a) Vous devez modifier la méthode qui saisit et valide l'option choisie par l'utilisateur" pour que la validation de l'option saisie se fasse entre 1 et 5 inclusivement.

   b) Vous devez ajouter une nouvelle méthode qui saisit et valide le nombre de véhicules à louer.

   - ✓ Cette méthode prend en paramètres le type et la grandeur du véhicule, et elle doit retourner le nombre de véhicules à louer.
   - ✓ Si le nombre de véhicules à louer est inférieur à 0 ou supérieur à 5, un message d'erreur est affiché, et la méthode doit demander à l'utilisateur d'entrer de nouveau un nombre entre 0 et 5 inclusivement.
   - ✓ Si le nombre de véhicules est entre 1 et 5 inclusivement, la méthode doit vérifier si le nombre de véhicules à louer est inférieur ou égal au nombre de véhicules disponibles dans l'inventaire (le même type, et la même grandeur), sinon un message d'erreur doit être affiché, et la méthode doit demander à l'utilisateur d'entrer un nouveau nombre de véhicules à louer. Elle doit faire appel à la méthode `estDisponible` de la classe `GestionVehiculesDisponibles` pour faire cette validation (le type et la grandeur passés en paramètres seront utilisés lors de cet appel). Elle doit également faire appel à

la méthode `obtenirNbVehiculesDisponibles` de la classe `GestionVehiculesDisponibles` pour afficher le message d'erreur.

c) Vous devez ajouter une nouvelle méthode qui saisit et valide la réponse de la question si le locataire désire louer un autre type et une grandeur de véhicule. Les réponses acceptées sont O ou o pour Oui, N ou n pour Non.

d) Vous devez modifier les méthodes qui saisissent l'option choisie par l'utilisateur, le nombre de jours de location, et le nombre de véhicules loués pour inclure la gestion des exceptions lorsque l'utilisateur entre un caractère autre qu'un entier.

2. Vous devez créer les classes `Locataire`, `Vehicule`, `VehiculeLoue`, `VehiculeDisponible`, `Facture`, et `LocationVehicule` comme décrites dans les tables ci-dessus.

3. Vous devez compléter les classes `ListeDesFactures`, `GestionVehiculesDisponibles`, `StatistiquesVehiculesLoues`, et `ApplicationPrincipale`.

4. Dans la méthode `main` de la classe `ApplicationPrincipale.java`, vous devez appeler les méthodes requises pour l'affichage du menu d'option, la saisie de l'option choisie par l'utilisateur, et appliquer les règles d'affaires selon l'option choisie. Lors de l'application de ces règles, vous devrez créer des objets et appeler les méthodes d'objets (voir la classe `ApplicationPrincipale.java` pour plus détails).

5. Toutes les classes à implémenter et à compléter doivent être créées dans un projet nommé `tp3`.

Notez-bien que vous pouvez ajouter toute autre classe ou méthode que vous jugez nécessaire pour faciliter votre travail.

4.2. Ce que vous devez remettre

La date de remise du travail pratique 3 est le **mercredi 23 avril 2025** avant **23H55**. Via Moodle, vous devez remettre une copie électronique du projet `tp3` compressée (.zip ou .rar) qui contient :

1. Les 10 classes (.java) : `Locataire`, `Vehicule`, `VehiculeLoue`, `VehiculeDisponible`, `LocationVehicule`, `Facture`, `ListeDesFactures`, `GestionVehiculesDisponibles`, `StatistiquesVehiculesLoues`, et `ApplicationPrincipale`.

2. Les 2 fichiers : `InventaireVehicules.csv` et `Factures.csv`.

Le travail pratique 3 peut se faire individuellement ou à un maximum de 2 étudiants. Si le travail est fait par 2 étudiants, une seule copie doit être remise par un des 2 étudiants dans le dossier de remise du travail pratique 3 sous Moodle. Assurez-vous de mettre les prénoms, les noms, et les codes permanents des 2 étudiants dans l'entête de chaque classe.

4.3. Pénalités

Pour tout travail remis en retard, les pénalités seront appliquées selon la formule suivante : `Nombre de points de pénalité = m / 144`, où m est le nombre de minutes de retard par rapport à l'heure de remise. Aucun travail ne sera accepté après cinq (5) jours de retard.

4.4. Plagiat

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues.

5. Pondération.

1. Code source (45) : le respect du nom de la classe, les noms significatifs pour les constantes, les variables et les méthodes, l'indentation des blocs d'instruction et aération, des messages d'invite clairs et précis, les bonnes structures de contrôle non redondantes, des commentaires pertinents, la définition des classes et leurs méthodes, la création des objets et les appels des méthodes, etc. sont des éléments qui seront évalués.
2. Exécution du programme (55) : la validation des saisies, l'application des règles d'affaires (respect des spécifications) et l'obtention des bons résultats, etc. sont des éléments qui seront évalués. La note zéro (0) sera attribuée à ce niveau pour tout programme Java qui ne compile pas ou tout programme Java qui NE contient PAS les différentes classes ci-dessus demandées.