Mission 1 : Retrait et restitution du véhicule loué

Question 1 : Ecrire le déclencheur *trgDateRetourLSC.* Pour cela, vous utiliserez la partie de la base de données entourée dans le document 1.

```
CREATE TRIGGER trgDateRetourLSC BEFORE INSERT ON LocationSansChauffeur
FOR EACH ROW
BEGIN
   /* déclaration d'une variable qui contiendra la durée de la location sans chauffeur
   DECLARE duree INTEGER ;
   DECLARE dateDepart DATETIME;
   DECLARE dateRetour DATETIME;

   /* récupération de la durée pour une  location sans chauffeur */
   SET duree = (SELECT duree FROM FormuleSansChauffeur
                WHERE id = NEW.idFormule );

   /* récupération de la date de départ dans une variable */
   SET dateDepart = (SELECT dateHreDepartPrevu FROM Location
       WHERE numLocation = NEW. numLocation) ;
   SET dateRetour  = DATE_ADD(dateDepart, INTERVAL duree HOUR) ;

   UPDATE Location set dateHreRetourPrevu = dateRetour
      where numLocation = NEW. numLocation ;
END ;
```

Question 2 : Écrire la requête répondant au besoin exprimé par votre chef de projet

```
SELECT M.nom, count(*)
FROM Vehicule V
JOIN Location L ON V.immatriculation = L.immatriculation
JOIN LocationSansChauffeur LSC ON LSC.numLocation = L.numLocation
JOIN Modele M ON V.idModele = M.id
GROUP BY M.id, M.nom,
ORDER by 2 desc;
```

Remarques :

- Le champ id dans le group by n'est pas exigé.
- On acceptera la syntaxe Mysql :
  select nom, count(*)
  from….join…
  group by M.id

Question 3 : Nombre de locations et montantRegle des locations de l'année 2009.

```
SELECT COUNT(*), SUM(montantRegle)
FROM Location
WHERE YEAR(dateLocation) = 2009
```

Question 4 : Afficher le nom de chaque client avec le nombre de locations. Liste triée dans l'ordre décroissant du nombre de locations.

```
SELECT c.id, c.nom, c.prenom, COUNT(*)
 FROM client c
 INNER JOIN location l  ON c.id = l.idClient
 GROUP BY 1
 ORDER BY 4 DESC
```

Question 5 : Même question avec les clients qui n'ont encore aucune location.

```
SELECT c.id, c.nom, c.prenom, COUNT(*)
 FROM client c
 LEFT JOIN location l  ON c.id = l.idClient
 GROUP BY 1
 ORDER BY 4 DESC
```

La requete suivante ne peut fonctionner car elle ne répond pas à la demande qui est affichée le nombre de locations par client et non le nom des clients sans location. Là cela sera O.

```
SELECT c.id, c.nom, c.prenom FROM client
WHERE c.id NOT IN (SELECT DISTINCT l.idClient
 FROM location l  ) ;
```

Question 6 : Donner la requête SQL permettant d'augmenter de 10% le prix du tarif du kilomètre supplémentaire dans la relation Modele.

```
UPDATE modele  SET tarifKmSupplementaire  = tarifKmSupplementaire  * 1.1;
```

Question 7 : Donner la requête SQL pour supprimer le client Durand

```
DELETE FROM location WHERE idClient  = (SELECT if FROM client WHERE nom = « Durand » AND prenom = « Pierre ») ;
 DELETE FROM client  WHERE nom = « Durand »
```

Question 8 : Créer en SQL une vue LocationYear qui pour chaque année, donne le chiffre d'affaires (cumul des montants réglès :

```
CREATE VIEW LocationYear AS
   SELECT YEAR(dateLocation), SUM(montantRegle) AS chiffreAffaires
   FROM location
GROUP BY 1
   ORDER BY 1
```

Question 9 : Créer un utilisateur ayant pour nom votre login et pour mot de passe "mdp"

```
CREATE USER "1234test " IDENTIFIED BY "mdp"
```

Question 10 : Donner à votre utilisateur les droits SELECT et UPDATE sur votre BDD

```
GRANT select, update ON LOCALUX.* TO 1234test;
```

Question 11 :Retirer le droit UPDATE sur votre base de données à cet utilisateur:

```
REVOKE update ON LOCALUX.* FROM 1234test ;
```

Question 12: Supprimer votre utilisateur

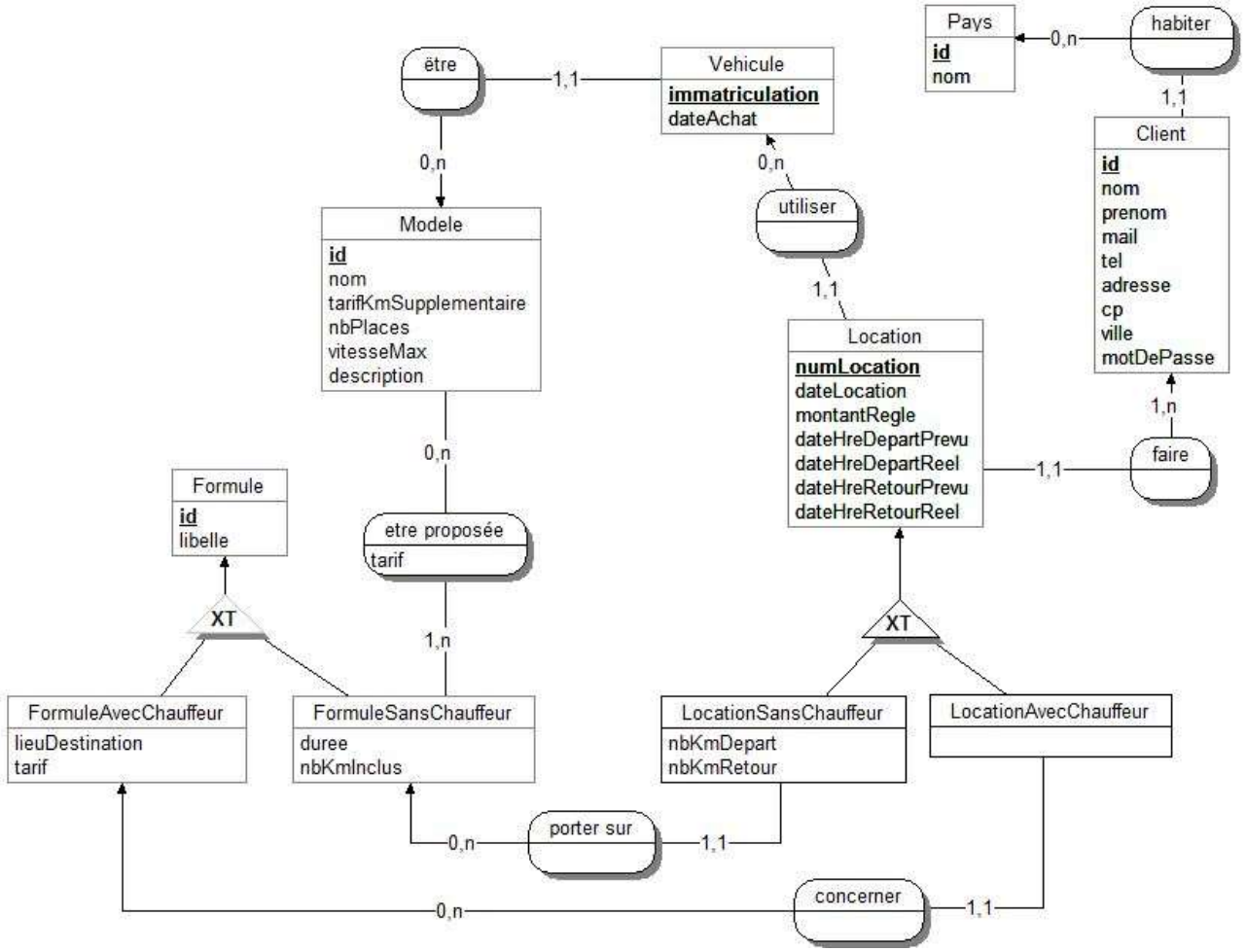
```
DROP USER 1234test ;
```

MCD MODELE CONCEPTUEL DE DONNEES

Question 13 : Fournir le MCD du document 1

Modélisation conceptuelle

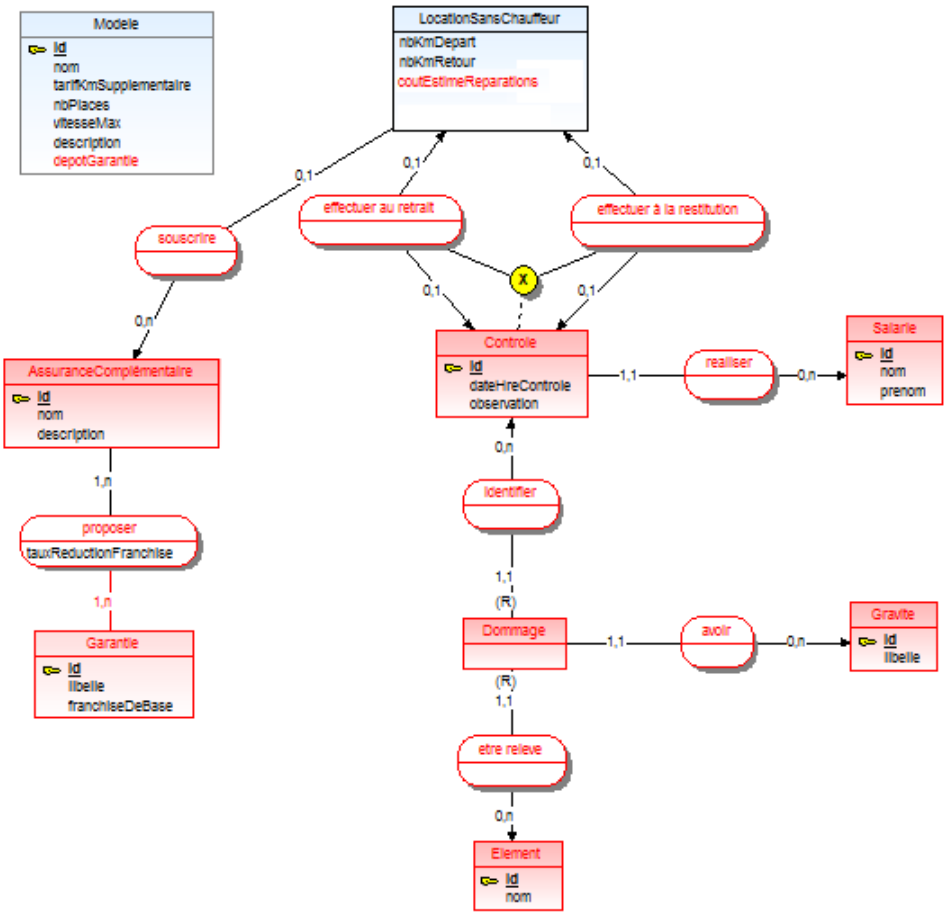
Schéma entité-association :



<u>Question 1.3</u> : Selon le formalisme de votre choix, compléter la structure de la base de données existante pour prendre en compte les données du nouveau module.

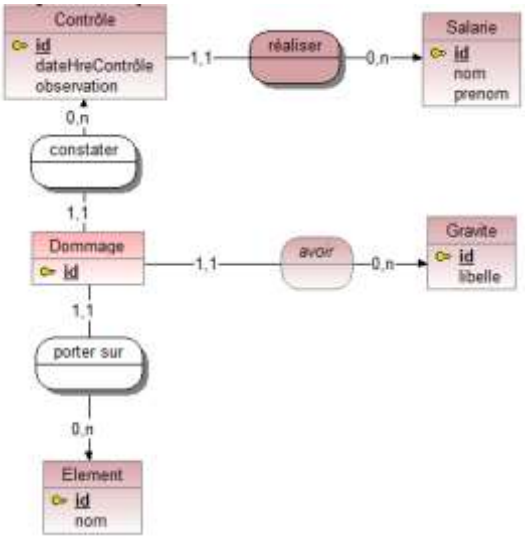
La modélisation conceptuelle est fournie sous les deux représentations les plus courantes.

<u>Schéma entité-association</u> :



Remarques pour la gestion des dommages :

- Au lieu de la pseudo-entité Dommage, on peut trouver une agrégation ou une association d'associations.
- On acceptera également cette solution pour l'entité Dommage (pas de lien identifiant):



Remarques pour la gestion des contrôles :

- La contrainte d'exclusion entre les 2 associations "*effectuer au retrait*" et "*effectuer à la restitution*" n'est pas exigée.
- On acceptera une spécialisation **Contrôle/ContrôleRetrait/ControleRestitution**
- Les 2 associations "*effectuer au retrait*" et "*effectuer à la réservation*" peuvent être remplacées par une association de type 0,2 – 1,1 entre LocationSansChauffeur et Contrôle.
- On pourra trouver, dans l'entité contrôle, une propriété indiquant si le contrôle a été fait avant ou après la location (cette propriété est non obligatoire car la dateHreControle permet de déterminer si le contrôle est un contrôle de retrait ou de restitution) :

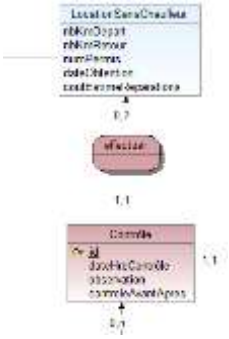


Diagramme de classes :

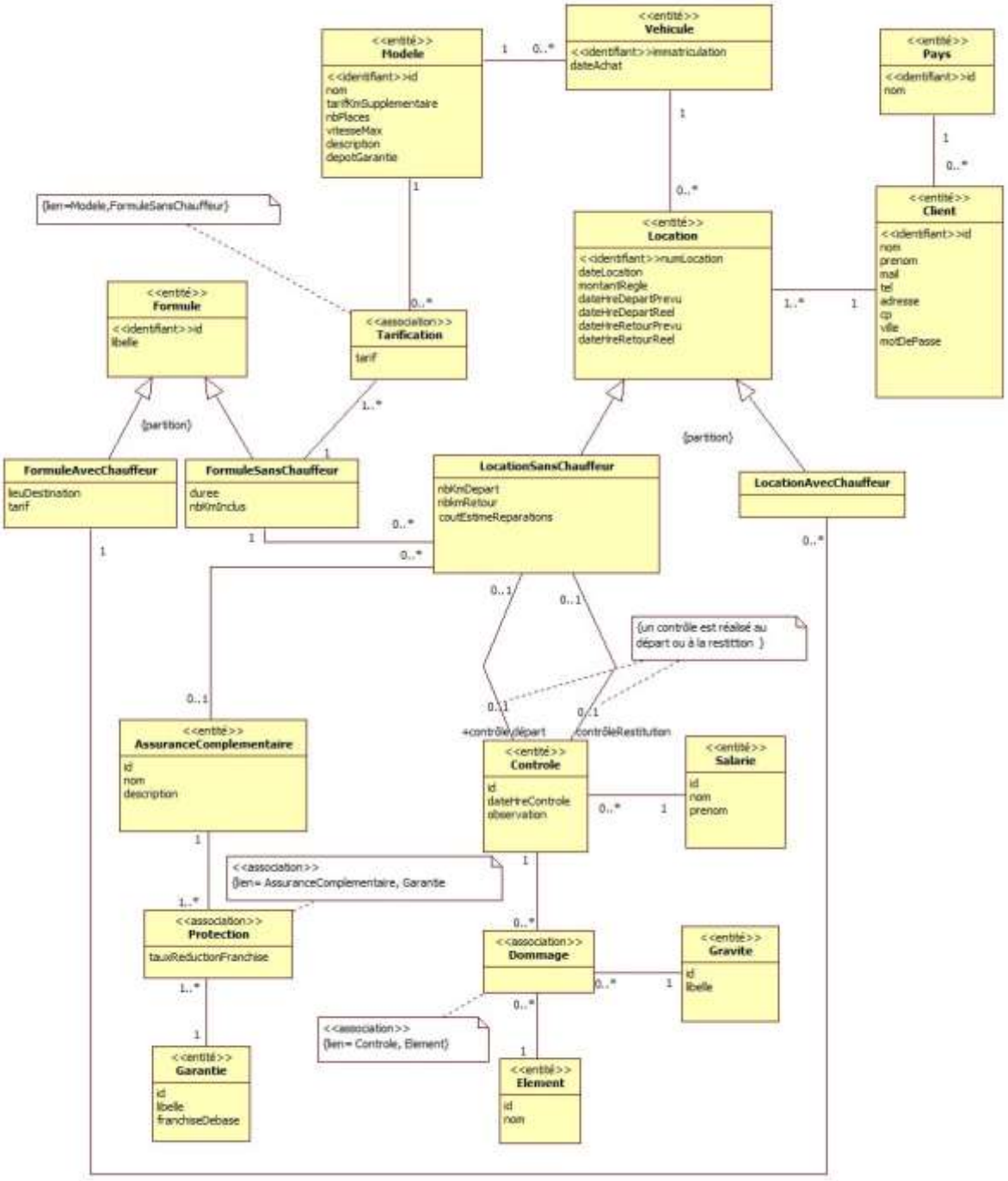

<<entité>>
Modele
<<identifiant>>id
nom
tarifKmSupplementaire
nbPlaces
vitesseMax
description
depotGarantie
<<entité>>
Vehicule
<<identifiant>>immatriculation
dateAchat
<<entité>>
Pays
<<identifiant>>id
nom
{lien=Modele,FormuleSansChauffeur}
<<entité>>
Location
<<identifiant>>numLocation
dateLocation
montantRegle
dateHreDepartPrevu
dateHreDepartReel
dateHreRetourPrevu
dateHreRetourReel
<<entité>>
Client
<<identifiant>>id
nom
prenom
mail
tel
adresse
cp
ville
motDePasse
<<entité>>
Formule
<<identifiant>>id
libelle
<<association>>
Tarification
tarif
{partition}
{partition}
FormuleAvecChauffeur
lieuDestination
tarif
FormuleSansChauffeur
duree
nbKmInclus
LocationSansChauffeur
nbKmDepart
nbKmRetour
coutEstimeReparations
LocationAvecChauffeur
{un contrôle est réalisé au départ ou à la restition }
+contrôleDépart
+contrôleRestitution
<<entité>>
AssuranceComplementaire
id
nom
description
<<entité>>
Controle
id
dateHreControle
observation
<<entité>>
Salarie
id
nom
prenom
<<association>>
{lien= AssuranceComplementaire, Garantie
<<association>>
Protection
tauxReductionFranchise
<<association>>
Dommage
<<entité>>
Gravite
id
libelle
<<association>>
{lien= Controle, Element}
<<entité>>
Garantie
id
libelle
franchiseDebase
<<entité>>
Element
id
nom

Schéma relationnel (modifications en rouge)

Modele (…, depotGarantie)

Location (…,coutEstiméReparations, idAssurance, idControleAvant, idControleApres)

Clés étrangères : *idAssurance en référence à id de* ***AssuranceComplémentaire***
idControleAvant en référence à id de Contrôle
idControleApres en référence à id de Controle

AssuranceComplementaire (id, nom, description)
Clé primaire : id

Garantie (id, libelle, franchiseDeBase)
Clé primaire : id

Propose (idAssurance, idGarantie, tauxReductionFranchise)
Clé primaire : idAssurance, idGarantie
Clés étrangères : idAssurance en référence à id de AssuranceComplementaire
idGarantie en référence à id de Garantie

Salarie (id, nom, prenom)
Clé primaire : id

Gravite (id, libelle)
Clé primaire : id

Element (id, nom)
Clé primaire : id

Controle(id, dateHreControle, observation, idLocation, idSalarie)
Clé primaire : id
Clés étrangères : idLocation en référence à id de Location
idSalarie en référence à id de Salarie

Dommage (idControle, idElement, idGravite)
Clé primaire : idControle, idElement
Clés étrangères : *idControle en référence à id de Controle*
idElement en référence à id de Element
idGravite en référence à id de Gravite

Mission 2 : Gestion du dépassement du forfait kilométrique

Question 2.1 : Réaliser le code du constructeur de la classe métier *LocationSansChauffeur.* Le kilométrage au compteur lors de la restitution du véhicule sera initialisé à 0.

```
public LocationSansChauffeur(int unNumLoc, DateTime uneDate ,
        double unMontantRegle, double unNbKmDepart, Vehicule unVehicule,
        FormuleSansChauffeur uneFormuleSansChauffeur) : base (unNumLoc,uneDate,
        unMontantRegle,unVehicule)
{
        this.nbKmDepart = unNbKmDepart ;
        this.nbKmRetour = 0 ;
        this.laFormuleAvecChauffeur = uneFormuleAvecChauffeur ;
 }
```

On pourra trouver une autre syntaxe pour appeler le constructeur de la classe mère (super en Java, parent en PHP, etc.).

Question 2.2 : Réaliser le code de la méthode *GetMontantDepasForfait()* de la classe métier LocationSansChauffeur qui permet d'obtenir le montant à régler par la personne cliente en cas de dépassement du forfait kilométrique.

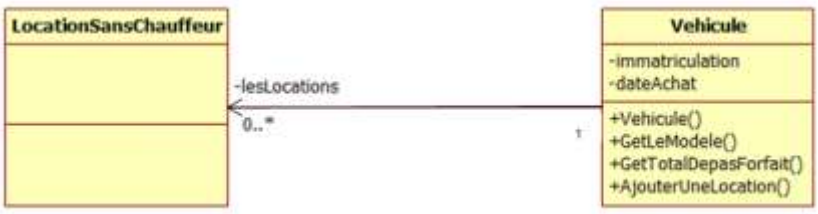
```
    public double GetMontantDepasForfait()
    {
        double montantDepasForfait;
        double kmDepass;
        double kmParcours;
        double prixKm;

        kmParcours = this.nbKmRetour - this.nbKmDepart;
        kmDepass = kmParcours - this.laFormuleSansChauffeur.GetNbKmInclus() ;
        montantDepasForfait = 0;

        if (kmDepass >0)
        {
                prixKm = this.GetLeVehicule().GetLeModele().GetTarifKmSupplementaire();
                montantDepasForfait = prixKm * kmDepass;
        }
        return montantDepasForfait;
    }
```

<u>Question 2.3</u> : Compléter le code du test unitaire de la méthode *GetMontantDepasForfait()*.

```
[TestMethod]
public void TestGetMontantDepasForfait()
{
    // création d'une instance de modèle
    Modele leModele = new Modele(1, "Renault Espace", 3);


    // création d'une instance de Vehicule
    Vehicule leVehicule = new Vehicule("LA-039-LP", new DateTime(2016, 12, 28),
                                                    leModele);
   // création d'une instance de FormuleSansChauffeur
    FormuleSansChauffeur laFormule = new FormuleSansChauffeur(1, "Forfait 24h", 24,300);
    // création d'une instance de LocationAvecChauffeur
    LocationSansChauffeur laLocation = new LocationSansChauffeur(1,
                            new DateTime(2017, 02, 15), 240, 25000, leVehicule, laFormule);

    // mise à jour du nombre de kilomètres au compteur
    laLocation.SetNbKmRetour(25400);

    //calcul du montant à régler pour la location
    double montantARegler = laLocation.GetMontantDepasForfait();
    Assert.AreEqual(300, montantARegler, "montant à régler pour le dépassement erroné");

}
```

<u>Question 2.4</u> : Afin de répondre au besoin de monsieur Berthu :

a) apporter les modifications nécessaires au diagramme de classes (*seules les classes concernées par les modifications devront être représentées sur votre copie*) ;



b) coder les modifications à apporter à la classe Véhicule ;

- ajout d'un attribut qui contient un ensemble d'instance de Location
- le constructeur doit être modifié : on ajoute une instruction pour créer la collection
- ajout de la méthode GetTotalDepasForfait
- ajout de la méthode AjouterLocation

```
// Attributs privés
   private  string immatriculation;
   private  DateTime dateAchat;
   private  Modele leModele;  // tarif du km supplémentaire

   private List<LocationSansChauffeur> lesLocations;


// Constructeur de la classe Vehicule
   public Vehicule(string uneImmat, DateTime uneDate, Modele unModele)
   {
         this.immatriculation = uneImmat;
         this.dateAchat = uneDate;
         this.leModele = unModele;
         this.lesLocations = new List<LocationSansChauffeur>();

    }

   public double GetTotalDepasForfait ()
   {
         double montantTotal= 0;
         foreach (LocationSansChauffeur uneLocation in this.lesLocations)
         {
                  montantTotal = montantTotal + uneLocation.GetMontantDepasForfait();
         }
         return montantTotal;
   }


    public void AjouterUneLocation(LocationSansChauffeur uneLocation)
   {
         this.lesLocations.Add(uneLocation);
    }
```

On pourra également accepter un diagramme UML (a) contenant une association entre Vehicule et **Location**.

Il y a alors 2 possibilités (selon l'implémentation retenue) pour le (b) :

- la classe Véhicule contient une collection de type Location
- mais la méthode GetTotalDepasForfait doit contenir un test permettant de connaitre le type d'une instance de la collection (utilisation de Typeof ou Instance of, etc.)

  ou

- la classe Véhicule contient une collection de type LocationSansChauffeur

Mission 3 : Choix de solution pour le développement d'une application mobile

Question 3.1 : Dresser un tableau comparatif, similaire à ceux utilisés chez DevApp, afin de comparer les différentes solutions de développement envisagées. Vous utiliserez les critères de comparaison suivants : temps de développement, ergonomie, rapidité d'exécution hors connexion réseau.

Solutions / Critères	Native	hybride	*web*
Temps de développement	*	**	***
Ergonomie	***	**	*
Rapidité d'exécution	***	**	*

*** * * Très satisfaisant** *** * Satisfaisant** *** Peu satisfaisant**

Pour un même critère, on acceptera deux étoiles pour 2 solutions différentes.

Question 3.2 : Détailler le calcul le montant total facturé à Localux.

Montant total facturé= 3 * 2 * 20 *400 = 48 k€

Question 3.3 : À partir de calculs chiffrés, déterminer la solution la moins coûteuse pour la première année.

Le reversement à Uber et Chauffeur Privé coûte 50 k€/an.
Le développement coûte 48€ plus 20k€ c'est-à-dire 68k€.
Le reversement coûte donc moins cher la 1ère année

Question 3.4 : Conclure sur l'évolution du coût de chaque solution au bout de trois ans.

- Solution n°1 : Au bout de la 3ème année, on a déjà reversé 150 k€ (50*3).
- Solution n°2 : Le développement d'une application par l'entreprise et son utilisation va coûter 48€ la 1ère année puis 20k€ les années suivantes. Au bout de la 3ème année, on aura payé 48 + 3*20 c'est-à-dire 108€ donc moins que le reversement sur 3 ans.

Cela implique que la 1ère solution qui était moins coûteuse la 1ère année, devient plus coûteuse à partir de la 2ème année. Comme il s'agit d'un investissement à long terme, on en conclut que le retour sur investissement est effectif au bout de la 3ème année ce qui permet de dire que la solution 2 est la solution à retenir en terme de stratégie informatique.

<u>Question 3.5</u> : Fournir d'autres éléments que le coût permettant d'opter pour l'une ou l'autre de ces deux solutions.

Confidentialité : difficulté à confier à un tiers une partie de son système d'information (l'application mobile constituant un des cœurs de métier de **Localux**)

Dépendance vis à vis d'une entreprise tiers peut être une contrainte (confiance, problèmes au sein de l'entreprise tiers etc…).

Perte de compétences puisque **Localux** délègue à un tiers la production du site mobile.

Moins bonne réactivité en cas d'incident

Possibilité de se concentrer sur son cœur de métier (la location de voitures)

Le fait de ne pas avoir à gérer le suivi des différentes applications (développement, mise à jour etc…) surtout si l'on n'a pas les compétences en interne.

TCO (dépenses directes et indirectes)