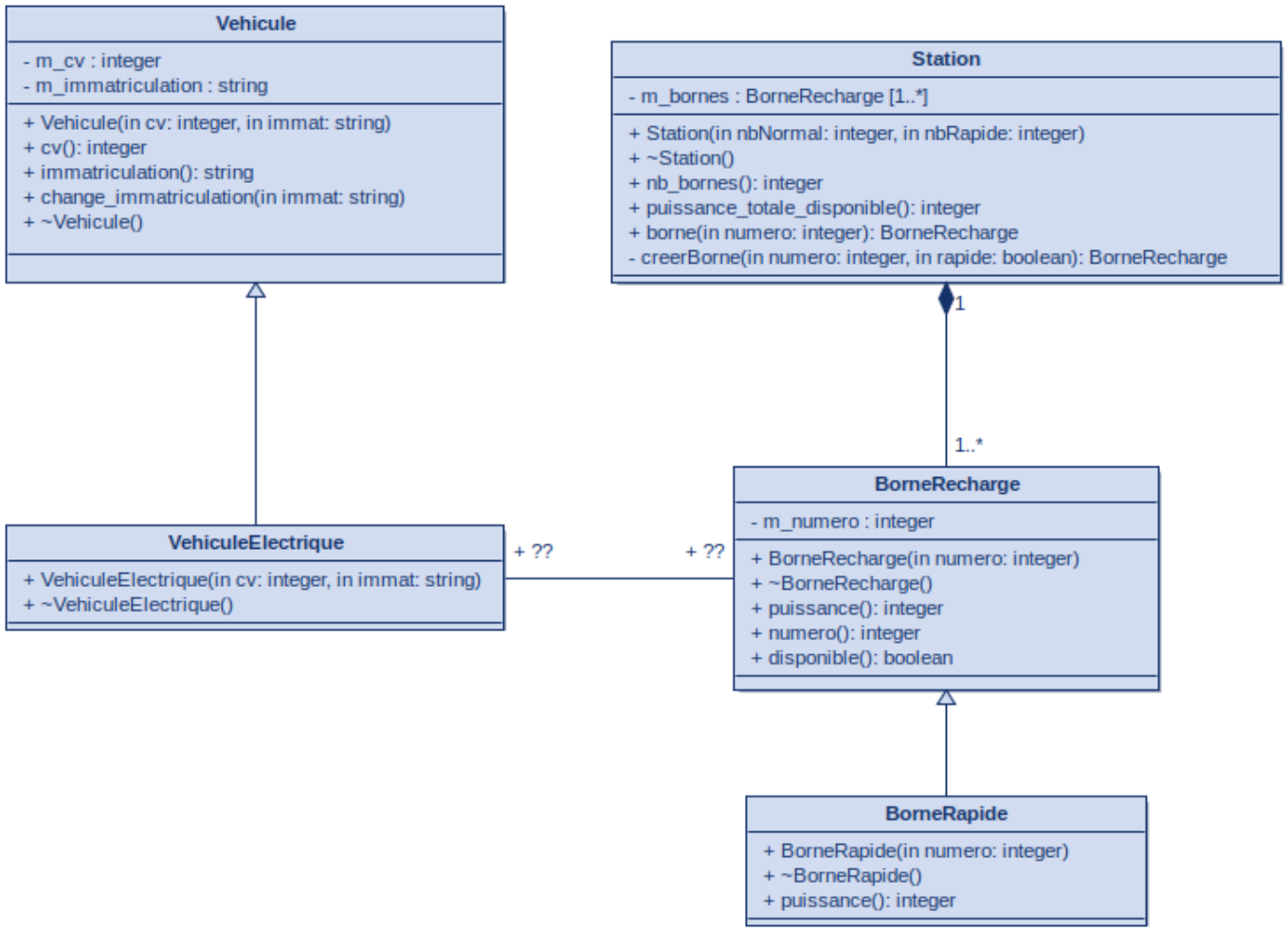
UE : Programmation Orientée Objet (I4ISIL11)
Sujet d'examen (Session 1: 8 avril 2025 10h)

Institut National des Sciences Appliquées de Toulouse
Yannick Pencolé, yannick.pencole@laas.fr

—

Durée : 1h15 – Tout document papier/électronique autorisé – Usage de compilateurs et assistants IA (chatGPT..) interdits

Toutes les questions sont indépendantes. Si vous bloquez à l'une d'entre elles, ne pas hésiter à répondre aux suivantes. Toutes les questions sont néanmoins en lien avec le diagramme de classes UML suivant. Le barême est à titre indicatif. Soignez votre rédaction.



Le schéma UML décrit un projet de gestion de station de recharge pour véhicules électriques. Chaque borne de recharge d'une station a une puissance de recharge et un numéro. Une borne de recharge est disponible si aucun véhicule électrique n'est branché sur la borne.

Partie 1 : Analyse du diagramme UML [environ 15min, 5pts]

Q1.1 : [0.5pt] Dans la classe `Vehicule`, combien y'a t-il d'attributs ? Sont-ils à accès publiques/privés/-protégés ?

Corrigé :
Deux attributs privés : `m_cv` et `m_immatriculation`.

Q1.2 : [0.5pt] Les attributs de la classe `Vehicule` ont-ils tous des accesseurs ? Quels sont ces accesseurs ?

Corrigé :
Oui. Deux accesseurs : `cv()` et `immatriculation()`.

Q1.3 : [1pts] Quel est le lien entre la classe `Vehicule` et la classe `VehiculeElectrique` ?

Corrigé :
C'est un lien d'héritage. `VehiculeElectrique` dérive de `Vehicule`.

Q1.4 : [3pts] Quel est le lien entre la classe `Station` et la classe `BorneRecharge`. Décrire les cardinalités. Quel est le mode d'accès de la méthode `creerBorne` ? Pourquoi selon vous ?

Corrigé :
C'est un lien de composition. Une `Station` est composée à minima d'une `BorneRecharge` et une `BorneRecharge` ne peut appartenir qu'à une `Station`. Comme il s'agit d'une composition la `Station` est en charge de la création et de la destruction des bornes qu'elle contient. C'est la raison pour laquelle il existe dans `Station` une méthode `creerBorne` d'accès privé. Aucun autre type d'objet n'a le droit de créer des bornes pour une station.

Partie 2 : Mise en œuvre [environ 60min, 15pts]

Rappel : en C++, les références à des objets se feront ici à l'aide de pointeurs. Même si cela n'est pas mentionné dans le diagramme UML, toute référence à un `Vehicule`, `VehiculeElectrique`, `Station`, `BorneRecharge`, `BorneRapide` est sous la forme d'un pointeur en C++. Au cours de cet examen, nous ferons abstraction de toute directive `#include<..>, #ifndef..., #using...`, ne pas les écrire.

Q2.1 : [3pts] Écrire une fonction `main` qui fait la séquence d'instructions suivantes :

1. Allocation dynamique d'un objet `v1` de type `Vehicule` de 4 cv immatriculé `"dc 234 eh"`.
2. Allocation statique d'un objet `v2` de type `VehiculeElectrique` de 5 cv immatriculé `"eg 364 vc"`.
3. Echanger l'immatriculation des deux véhicules.

Que faut-il ajouter en fin du programme `main` pour que ce programme ne produise pas de fuite de mémoire ?

Corrigé :

```
int main() {
  Vehicule * v1 = new Vehicule(4,"dc 234 eh");
```

```
    VehiculeElectrique v2(5,"eg 364 vc");
    string immat = v1->immatriculation();
    v1->change_immatriculation(v2.immatriculation());
    v2.change_immatriculation(immat);
    delete v1;
  }
```

Q2.2 : [2pts] Écrire en C++ la spécification de la classe `Vehicule`. On ne demande pas sa mise en œuvre.

Corrigé :

```
  class Vehicule {
    private:
    unsigned m_cv;
    string m_immatriculation;
    public:
    Vehicule(unsigned cv, string immatriculation);
    unsigned cv();
    string immatriculation();
    void change_immatriculation(string immat);
    ~Vehicule();
  };
```

Q2.3 : [1pt] Écrire en C++ la mise en œuvre du constructeur de `VehiculeElectrique`. On ne demande pas de spécification de la classe `VehiculeElectrique`.

Corrigé :

```
  VehiculeElectrique::VehiculeElectrique(unsigned cv, string immatriculation):
     Vehicule(cv,immatriculation){}
```

Q2.4 : [3pts] Dans la classe `Station`, chaque borne est numérotée entre `0` et `nb_bornes()-1`. En utilisant exclusivement les méthodes à disposition dans les classes `Station` et `BorneRecharge` (pas les attributs), mettre en œuvre la méthode `puissance_totale_disponible()` de la classe `Station` qui retourne la somme des puissances des bornes disponibles de la station. Seule la méthode `puissance_totale_disponible()` est demandée, les autres seront supposées connues.

Corrigé :

```
  unsigned Station::puissance_totale_disponible()
  {
    unsigned result = 0;
    for(unsigned i = 0; i < nb_bornes(); ++i)
    {
      if(borne(i)->disponible())
```

```
      {
        result += borne(i)->puissance();
      }
    }
    return result;
  }
```

Q2.5 : [3pts] Une borne de recharge classique retourne une puissance de 50kW (méthode `puissance()`). Nous souhaitons spécialiser la méthode `puissance()` dans la classe `BorneRapide` pour qu'elle retourne 350kW. Détailler les éléments de mise en œuvre de la méthode `puissance()` dans la classe `BorneRapide`.

Corrigé :
Mise en œuvre de `puissance()` dans `BorneRapide`.

```
unsigned BorneRapide::puissance()
{
  return 350;
}
```

Il faut également virtualiser les méthodes `puissance()`. Ainsi dans la classe `BorneRecharge` et dans la classe `BorneRapide` on ajoute :

```
  virtual unsigned puissance();
```

et on virtualise les destructeurs : `BorneRecharge`

```
  virtual ~BorneRecharge();
```

et celui de `BorneRapide`

```
  virtual ~BorneRapide();
```

Q2.6 : [3pts] Un véhicule électrique peut venir se brancher sur une borne de station si elle est disponible. Une fois branché, la borne en question devient indisponible. On peut ensuite débrancher le véhicule électrique et la borne redevient disponible. Proposer une mise en œuvre de cette association entre `VehiculeElectrique` et `BorneRecharge` (nouveaux attributs ? nouvelles méthodes ?). Quelles sont les cardinalités de cette association ?

Corrigé :
Le descriptif ci-dessus détaille une association entre une `BorneRecharge` et un `VehiculeElectrique`. Les cardinalités sont donc de 1 des deux côtés de l'association. La borne de recharge joue le rôle de `borne` pour le `VehiculeElectrique`, et le `VehiculeElectrique` joue le rôle de `vehicule` pour la `BorneRecharge`. Voiçi une proposition de mise en œuvre.

1. On créé un attribut `BorneRecharge * m_borne;` dans `VehiculeElectrique` ;
2. on créé un attribut `VehiculeElectrique * m_vehicule;` dans `BorneRecharge` ;
3. on met en œuvre les méthodes
   — `bool brancher(BorneRecharge * borne); bool debrancher();` dans la classe `VehiculeElectrique` ;
   — et les méthodes `bool brancher(VehiculeElectrique * vehicule); bool debrancher();` dans la classe `BorneRecharge`.

```
bool VehiculeElectrique::brancher(BorneRecharge * borne)
{
  if(!borne->disponible())
  {
    return false;
  }
  m_borne = borne;
  if(borne->brancher(*this))
  {
    return true;
  }
  else
  {
    m_borne = nullptr;
    return false;
  }
}

bool VehiculeElectrique::debrancher()
{
  if(m_borne != nullptr)
  {
    m_borne->debrancher();
    m_borne=nullptr;
    return true;
  }
  else
  {
    return false;
  }
}

bool BorneRecharge::brancher(VehiculeElectrique * vehicule)
{
  if(disponible())
  {
    m_vehicule = vehicule;
    return true;
  }
  else
  {
    return false;
  }
}
bool BorneRecharge::debrancher()
{
  if(!disponible())
  {
    m_vehicule = nullptr;
    return true;
  }
  else
  {
    return false;
  }
}
bool BorneRecharge::disponible()
{
  return m_vehicule == nullptr;
```

```
    }
```