Université de Nice-Sophia Antipolis — POLYTECH
ELSE4-FISE — 2025–2026

Examen de C++

Durée : 1h00
Aucun document autorisé
Mobiles interdits

La société *RouleElec* est une société de location de véhicules électriques. Elle loue des voitures et des utilitaires et souhaite informatiser la gestion de sa flotte pour calculer la consommation électrique de ses véhicules.

▶ 1. Dans le fichier `véhicule.hpp`, écrivez la classe *abstraite* `véhicule` qui possède :
- les 2 variables membres protégées, `marque` et `capacitéBatterieKWh`, qui sont, respectivement, la marque du véhicule (type `std::string`) et la capacité de la batterie en kwh (type **double**) ;
- un constructeur pour initialiser les variables membres ;
- le destructeur ;
- la méthode virtuelle *pure* `calculerConsommationParKm()` qui calcule la consommation par km du véhicule ;
- la méthode virtuelle `toString()` ;
- la méthode `calculerAutonomieMax()` qui renvoie l'autonomie maximale du véhicule.
- l'opérateur `<<` qui écrit dans un `ostream` un `véhicule`.

Vous n'écrirez que les en-têtes des méthodes.

Question sur 3 pts

```
class véhicule {
protected:
    std::string marque;
    double capacitéBatterieKWh;

public:
  véhicule(const std::string& m, const int capacité);
  virtual ~véhicule() =default;
  virtual double calculerConsommationParKm() const = 0;
  double calculerAutonomieMax() const;
  virtual std::string toString() const;
  friend std::ostream & operator<<(std::ostream &f, const véhicule &v);
};
```

.....

▶ 2. L'autonomie maximale d'un véhicule est la capacité de la batterie divisée par sa consommation par km. Est-il possible de programmer la méthode `calculerAutonomieMax` ? Expliquez.
Si oui, écrivez cette fonction dans le fichier `véhicule.cpp`.

Question sur 3 pts

Oui bien sûr, elle utilise la méthode virtuelle pure **this->**`calculerConsommationParKm()` ce qui mettra en jeu la liaison dynamique.


```
double véhicule::calculerAutonomieMax() const {
  double consommation = this->calculerConsommationParKm();
  if (consommation > 0.0)
    return this->capacitéBatterieKWh / consommation;
  // else
  return 0.0;
}
```

.....

▶ 3. Dans le fichier `voiture.hpp`, écrivez la classe `voiture` héritière de `véhicule` qui possède une variable membre privée `nombreDePlaces`, son constructeur, la méthode `calculerConsommationParKm` et la méthode `toString`. *N'écrivez que les en-têtes.*

Question sur 2 pts

```
class voiture : public véhicule {
private:
    int nombreDePlaces;
public:
  voiture(const std::string& m, const int capacité, const int nbP);
  virtual double calculerConsommationParKm() const override;
  virtual std::string toString() const override;
};
```

.....

▶ 4. Dans le fichier `voiture.cpp`, écrivez la méthode `calculerConsommationParKm`. Une voiture possède une consommation de base de 0.15 kwh, plus 0,01 kwh par km pour chaque place au delà de 2. Par exemple, une voiture 4 places consomme $0.15 + (2 \times 0.01) = 0.17$ kwh/km.

Question sur 2 pts

```
double voiture::calculerConsommationParKm() const {
  double consoBase=0.15;
  if (this->nombreDePlaces>2)
    return consoBase + 0.01*(this->nombreDePlaces-2);
  //
  return consoBase;
}
```

.....

On souhaite gérer la flotte des véhicules de la société de location *RouleElec* à l'aide la classe *générique* `flotte`. Cette classe sera paramétrée avec le type générique `T`. Par la suite, on considère que la classe `utilitaire`, définie sur le même modèle que la classe `voiture`, est disponible. Toutefois, à la place du nombre de places, elle mémorise le volume utile en mètres cubes. La classe `flotte` contient :
- une variable membre privée `lesVéhicules` de type `std::vector<T>` ;
- la méthode **void** `ajouterVéhicule(const T v)` qui ajoute un véhicule dans le vecteur ;
- la méthode `toString()` qui renvoie une `std::string` qui décrit la flotte courante ;
- l'opérateur `<<` qui écrit dans un `ostream` une `flotte`.

▶ 5. Peut-on écrire les méthodes de la classe `flotte` dans un fichier `.cpp` ? Expliquez.

Question sur 1 pt

Non car c'est une classe générique.

.....

▶ 6. Écrivez la classe `flotte`.

Question sur 4 pts

```
template <typename T>
class flotte {
private:
  std::vector<T> lesVéhicules;
public:
  void ajouterVéhicule(const T &v) {
    assert(v!=nullptr);
    this->lesVéhicules.push_back(v);
  }
  //
  std::string toString() const {
    std::string s;
    for (const T &v : this->lesVéhicules)
      s+=v->toString();
    return s;
  }
  //
  friend std::ostream & operator<<(std::ostream &f, const flotte<T> &fl) {
    return f << fl.toString();
  }
};
```

.....

▶ 7. Dans la fonction `main`, on veut créer une flotte avec des véhicules *alloués dynamiquement*. En l'absence de destructeur spécifique dans la classe `flotte`, donnez la déclaration d'une variable `maFlotte` pour des objets `véhicule` alloués dynamiquement. Expliquez.

Question sur 2 pts

En l'absence de destructeur spécifique dans la classe `flotte`, si les éléments du vecteur ont été alloués dynamiquement, il faut utiliser des *smart pointers* pour qu'ils soient détruits automatiquement à la destruction de la variable `maflotte`. On déclare :

```
flotte<std::shared_ptr<véhicule>> maFlotte;
```

.....

▶ 8. La classe `CompteBancaire` possède une variable membre privée `solde` (type **double** $\geqslant 0.0$), et un constructeur pour initialiser le solde. Écrivez la méthode **void** `retirer(const double m)` qui fait un retrait du montant `m` sur le compte bancaire courant. Cette méthode lèvera l'exception `ExceptionMontantInvalide` si `m` $\leqslant 0$ et l'exception `ExceptionSoldeInsuffisant`. Ces deux exceptions héritent de `std::exception`. *Vous n'avez pas à écrire ces deux classes d'exception.*

Question sur 2 pts

```
// Antécédent : montant>0
void CompteBancaire::retirer(const double montant) {
  if (montant <= 0.0)
    throw ExceptionMontantInvalide(montant);
  //
  if (this->solde < montant)
      throw ExceptionSoldeInsuffisant(this->solde, montant);
  // montant valide pour le retrait
  this->solde -= montant;
}
```

.....

▶ 9. Écrivez le fragment de code qui déclare la variable `cpt` de type `CompteBancaire` avec 2 000 euros de solde, qui lit sur l'entrée standard un montant à retirer, et gère la capture des deux exceptions. Vous écrirez sur `std::cerr` le message fourni par `what` pour chacune des deux exceptions.

Question sur 2 pts

```
CompteBancaire cpt(2000);
try {
    double m;
    std::cin >> m;
    cpt.retirer(m);
    // retrait validé
}
catch (const ExceptionMontantInvalide &e) {
  std::cerr << e.what() << std::endl;
}
catch (const ExceptionSoldeInsuffisant &e) {
  std::cerr << e.what() << std::endl;
}
```

.....