Exercice I :

Répondez aux questions suivantes (2 points par bonne réponse : 1 point pour la réponse et 1 point pour la justification).

Question 1

```
public class A {
    private int i;
    public int getI() {
        return i;
    }
}
public class B : A {
    public int getI() {
        return base.getI() + 2;
    }
}
class Test {
    public static void Main(String[] args) {
        B a = new A();
        Console.WriteLine(a.getI());
        Console.WriteLine(((A)a).getI());
        Console.WriteLine(((B)a).getI());
        Console.ReadLine();
    }
}
```

Réponses possibles :

- A1. La classe Test ne compile pas. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- B1. La classe Test compile mais provoque une erreur à l'exécution. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- C1. La classe Test compile et affiche ..... (indiquez ce qui sera affiché et pourquoi).

Question 2

Dans la question précédente on modifie la ligne

```
B a = new A();
```

en

```
A a = new B();
```

Réponses possibles :

- A2. La classe Test ne compile pas. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- B2. La classe Test compile mais provoque une erreur à l'exécution. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- C2. La classe Test compile et affiche ..... (indiquez ce qui sera affiché et pourquoi).

Question 3

```
public class A {
    private int i;
    public virtual int getI() {
        return i;
    }
}
public class B : A {
    public override int getI() {
        return base.getI() + 2;
    }
}
class Test {
    public static void Main(String[] args) {
        A a = new B();
        Console.WriteLine(a.getI());
        Console.WriteLine(((A)a).getI());
        Console.WriteLine(((B)a).getI());
        Console.ReadLine();
    }
}
```

Réponses possibles :

- A3. La classe Test ne compile pas. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- B3. La classe Test compile mais provoque une erreur à l'exécution. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- C3. La classe Test compile et affiche ..... (indiquez ce qui sera affiché et pourquoi).

Question 4

```
public class A {
    private int i;
    public A(int i)
    {
        this.i = i;
    }
    public virtual int getI() {
        return i;
    }
}
public class B : A {
    public override int getI() {
        return base.getI() + 2;
    }
}
class Test {
    public static void Main(String[] args) {
        A a = new A(5);
        Console.WriteLine(a.getI());
        Console.WriteLine(((A)a).getI());
        Console.WriteLine(((B)a).getI());
        Console.ReadLine();
    }
}
```

Réponses possibles :

- A4. La classe Test ne compile pas. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- B4. La classe Test compile mais provoque une erreur à l'exécution. Indiquez la ligne qui provoquera l'erreur et dire pourquoi.
- C4. La classe Test compile et affiche ..... (indiquez ce qui sera affiché et pourquoi).

Exercice II :

Vous allez écrire une application pour gérer la répartition des charges dans des immeubles.
La société pour laquelle vous allez écrire cette application gère plusieurs immeubles. Chaque immeuble contient des appartements, des garages et des caves qui appartiennent à des propriétaires, et des parties communes qui appartiennent à la copropriété (c'est-à-dire qui appartiennent à tous les propriétaires). Dans les parties communes on peut trouver les voies d'accès aux appartements, les couloirs, l'entrée de l'immeuble, les ascenseurs, etc. On ne vous demande pas de modéliser les parties communes.
Chaque partie de l'immeuble qui appartient à un propriétaire s'appelle un **lot**. Chaque lot a un numéro qui l'identifie dans l'immeuble. Il y aura ainsi le lot 1, le lot 2, le lot 3, etc. Chaque lot a un certain nombre de tantièmes. Ce nombre de tantièmes permettra de calculer les charges dues par le propriétaire du lot. Par exemple si le lot 1 a 230 tantièmes, si le total de tous les tantièmes pour l'immeuble est 1000 et que les charges totales à payer sont de 2000 euros, le propriétaire du lot 1 devra payer (230 / 1000) x 2000 = 460 euros.
Il y a 3 sortes de lots :

- les appartements ;
- les garages ;
- les caves.

Voici les règles utilisées pour le calcul des tantièmes (ces règles ont été simplifiées pour les besoins de l'exercice et ne reflètent pas exactement les règles utilisées dans la réalité) :

- On part de la superficie (la surface) du lot ; prenons l'exemple du lot 8 qui est un appartement de 100 m², situé au 2ème étage, qui a un balcon de 10 m².

- On lui applique un coefficient multiplicateur qui dépend de sa nature : 3 pour les appartements, 2 pour les garages fermés et 1 pour les caves, les garages non fermés ou les balcons ; pour notre exemple, cela fait 100 x 3 + 1 x 10 = 310.
- On applique ensuite un autre coefficient multiplicateur qui dépend de l'étage : 10 pour le rez-de-chaussée, 14 pour le premier, 18 pour le deuxième étage, etc. (on ajoute 4 par étage). Pour notre exemple, on arrive à la valeur finale de 310 x 18 = 5580 tantièmes pour le lot 8. Les caves et les garages ont ce coefficient multiplicateur égal à 14.

Autre exemple de calcul : le lot 34 est une cave de 10 m² ; son nombre de tantièmes final est de 10 x 1 x 14 = 140 tantièmes.
Si les charges totales pour tout l'immeuble se montent à 10000 euros et que le total des tantièmes pour tous les lots se monte à 100000, le propriétaire du lot 8 devra payer 10000 x 5580 / 100000 = 558 euros et le propriétaire du lot 34 devra payer 10000 x 140 / 100000 = 14 euros.
Vous allez modéliser les lots d'un immeuble avec des classes C#. Vous écrirez une classe pour les immeubles, une classe pour les appartements, une classe pour les garages et une classe pour les caves. Vous aurez peut-être aussi à écrire d'autres classes.
Informations pour un immeuble :

- adresse de l'immeuble, une chaîne de caractères ;
- les lots qui appartiennent à l'immeuble (un **tableau** de lots ; interdit d'utiliser une collection pour ceux qui savent ce que c'est).

Voici les informations qui doivent être associées aux différents lots.
Pour les appartements :

- numéro de lot, un nombre entier ;
- surface en m², un nombre entier ;
- nom du propriétaire, une chaîne de caractères ;
- étage, un nombre entier (0 pour le rez-de-chaussée, 1 pour le premier étage,...) ;
- surface totale des balcons en m², un nombre entier.

Pour les garages :

- numéro de lot, un nombre entier ;
- surface en m², un nombre entier ;
- nom du propriétaire, une chaîne de caractères ;
- si le garage est fermé ou pas, un booléen ; un garage non fermé correspond à une place de parking.

Pour les caves :

- numéro de lot, un nombre entier ;
- surface en m², un nombre entier ;
- nom du propriétaire, une chaîne de caractères.

Question 1
Dans cette question on ne s'occupe pas des tantièmes.
Donnez le code de classes pour modéliser les immeubles, les appartements, les garages et les caves. Pour vous éviter d'avoir trop de code à écrire on ne vous demande pas d'écrire le code des propriétés (accesseurs et des modificateurs), **à part celles qui sont associées au propriétaire** et **celles qui permettent d'accéder au numéro et à la surface d'un lot**.
De même, écrivez seulement un **constructeur dans la classe qui représente les immeubles** : le constructeur prend en paramètre l'adresse de l'immeuble ainsi que le nombre maximum de lots que l'immeuble contiendra. Les lots seront ajoutés ensuite par une méthode (voir question 2).
Ecrivez aussi le **constructeur de la classe associée aux garages**, celui qui initialise toutes les variables d'instance. Vous aurez peut-être aussi à écrire du code dans d'autres classes pour que ce constructeur fonctionne.
Pour vous éviter d'avoir trop de code à écrire on ne vous demande pas de donner le code des autres constructeurs.
On ne vous demande pas d'écrire le code des méthodes `ToString`().

Question 2
Dans la classe qui correspond à un immeuble vous écrirez une méthode `ajouterLot` qui permet d'ajouter un nouveau lot passé en paramètre ; vous supposerez pour simplifier votre code que le lot n'a pas déjà été ajouté.

Question 3
Vous écrirez aussi une méthode `supprimerLot` pour supprimer un lot. Cette dernière méthode prendra en paramètre le numéro du lot ; elle ne fera rien si le numéro du lot n'existe pas mais, dans ce cas, elle renverra la valeur booléenne `false` (elle renverra `true` si le lot existe).

Question 4
Vous allez modifier le code déjà écrit pour calculer les tantièmes de tous les lots d'un immeuble.

Ecrivez une méthode `calculerTantiemes` dans la classe associée aux immeubles, qui lancera le calcul de ces tantièmes pour tous les lots de l'immeuble. Vous donnerez aussi le code des autres méthodes éventuelles que vous devrez ajouter aux classes déjà écrites pour faire fonctionner cette méthode `calculerTantiemes`. Pensez à encapsuler au maximum vos classes et à bien choisir la classe qui va calculer les tantièmes d'un lot.
Les tantièmes calculés par la méthode devront être conservés dans des variables d'instance. Par exemple, les tantièmes d'un garage devront être enregistrés dans l'instance qui représente le garage.
Ne récrivez pas tout le code ; indiquez seulement **très clairement** le nouveau code, ainsi que l'endroit où ce code devra être ajouté.
Expliquez où vous avez utilisé le polymorphisme.

Exercice 3 : On définit les deux classes suivantes :

```
public class Shape
{
    private double x, y;
    public Shape(double x, double y)
    {
        this.x = x; this.y = y;
    }
    public string toString()
    {
        return "Position : (" + x + "," + y + ")";
    }
    public Shape()
    {
        x = 0; y = 0;
    }
}

class Circle : Shape
{
    static double PI = 3.141592564;
    private double radius;
    public Circle()
    {
        radius = 0;
    }
    public Circle(double x, double y, double r)
        : base(x, y)
    {
        radius = r;
    }
}
```

Le programme principal contient les lignes suivantes :

```
Circle c1,c2 ;
c1 = new Circle(1,1,3) ;
c2 = new Circle() ;
Console.WriteLine(c1.toString() + "\n" + c2.toString()) ;
Console.ReadLine();
```

1. Etendre la méthode `toString` à la classe `Circle`.
2. De quelles variables d'instance de `Shape` hérite la classe `Circle` ?
3. La variable radius étant déclarée `private`, on ne peut pas la modifier de l'extérieur de la classe. Ajoutez une propriété (`Radius`) pour pouvoir la lire et l'écrire.
4. Ajoutez une variable `surface` à la classe `Circle`. Modifiez la propriété `Radius` et la méthode `toString` en conséquence. Ajoutez la propriété pour accéder à ce champ.

5. Écrivez une méthode `c1.isBigger(Circle c2)` renvoyant ``vrai'' si le cercle c1 est plus grand que c2. Comment écrire une méthode ayant un comportement similaire, mais prenant les deux cercles en argument ?
6. Étendez la classe `Circle` avec une classe `Cylinder` ajoutant une variable h pour la hauteur et une méthode `volume`. On ajoutera les constructeurs adéquats et on réécrira `toString` en conséquence.
7. Étendez la classe `Circle` avec une classe `ColoredCircle`. La classe `Cylinder` hérite t-elle de l'attribut de coloration ?
8. Créer un constructeur pour la classe `cylinder` qui prend comme argument un cercle et un rayon.
9. Surchargez l'opérateur « + » pour pouvoir additionner deux `Shape`.

Exercice4 : Transport de marchandises

On souhaite modéliser en C# le calcul de coûts de transport de marchandises. Les marchandises transportées seront des instances de la classe Marchandise :

```
class Marchandise
{
    private string nom;
    private int poids;
    private int volume;
    public Marchandise(int P, int V, string N)
    {
        this.poids = P;
        this.volume = V;
        this.nom = N;
    }
    public int Poids
    {
        get { return poids; }
    }
    public int Volume
    {
        get { return volume; }
    }
    public string Nom
    {
        get { return nom; }
    }
}
```

Les marchandises sont transportées sous la forme de cargaisons. Les seules fonctionnalités publiques des cargaisons sont :

- `AjouterMarchandise` qui permet d'ajouter une marchandise dans cette cargaison si cela est encore possible.
- `Cout` qui retourne, sous la forme d'un nombre entier d'euros, le coût total du transport de cette cargaison.

Une cargaison est par ailleurs également caractérisée par la distance sur laquelle elle est transportée. Ce renseignement est communiqué à la construction de la cargaison sous la forme d'un nombre entier de kilomètres. On précise qu'une cargaison ne peut réunir qu'un nombre limité de marchandises qui dépend d'un encombrement total de ces marchandises à ne pas dépasser. Cet encombrement est soit le poids total, soit le volume total des marchandises, selon le type de transport utilisé. Ce dernier influe aussi sur le calcul du coût de transport de la cargaison qui, de la même façon, dépend de l'encombrement des marchandises de la cargaison. On distingue donc plusieurs types de cargaisons selon le moyen de transport utilisé. On peut cependant trouver un certain nombre de caractéristiques communes `a toutes les cargaisons que vous

devrez identifier. Les différents types de cargaison et leurs caractéristiques sont donnés par le tableau suivant :

Type	Encombrement	Coût	Limite
Fluviale	Poids	Distance x Encombrement	Encombrement < 300000
Routière	Poids	2 xDistance x Encombrement	Encombrement < 38000
Aérienne	Volume	4x Distance x Encombrement	Encombrement < 80000
Aérienne urgente	Volume	2 x le coût d'une cargaison aérienne	Encombrement < 80000

Implémentez ces différentes classes (on utilisera une classe abstraite).