Objectifs :

- Analyser un problème en classes et en sous-classes.
- Mettre en œuvre les notions d'attributs, de méthodes, mais aussi de visibilité (ou accessibilité) et d'héritage.
- Initiation au langage Java.

Exercice – « parc de véhicules »

Nous désirons développer un programme pour la gestion (très simplifiée) d'un parc de véhicules destinés à la location.

Pour chaque véhicule, il est demandé de renseigner le *modèle*, l'*année d'achat*[1], le *prix d'achat*, le *numéro d'immatriculation* et le *permis* (une lettre) nécessaire à la conduite de ce véhicule.

Plusieurs types de véhicules sont possibles à la location, certains ayant des caractéristiques particulières. Ainsi, il est demandé de représenter l'information comme quoi une voiture dispose ou non d'un *autoradio*. Pour la location d'un camion, le *volume* de stockage possible du camion doit être précisé. Enfin, des autocars peuvent aussi être loués, et pour ces véhicules il est nécessaire de connaître leurs *volumes* de stockage possibles ainsi que le *nombre de passagers* qu'ils peuvent transporter.

1. Modélisation

1. Dessiner l'arbre d'héritage de classes, où figurera les noms des classes, les attributs et leurs types, et les liens d'héritage.
2. Traduire la définition des classes en Java. Prévoyez un constructeur pertinent pour chaque classe.
3. Instancier dans un programme principal (méthode "main" d'une classe, la classe *Test* par exemple) chacune de ces classes.

2. Codage des méthodes

1. Écrire les méthodes *ajouterAutoradio()* et *enleverAutoradio()* qui permettent de fixer la valeur de l'attribut *autoradio* d'une voiture.
2. Écrire la méthode *age()* qui retourne l'âge d'un véhicule (en nombre d'année).
3. Écrire la méthode infoVoiture*()* qui retourne une chaîne de caractères (*String)* caractérisant une voiture.
4. Écrire une méthode *peutTransporterVolume(...)* qui à partir d'un volume donné en paramètre retourne vrai ou faux selon que le véhicule peut ou non permettre le transport de ce volume[2].
5. Écrire une méthode coutLocation*(...)* qui calcule le coût quotidien de location d'un véhicule. Ce coût est calculé comme suit : pour les véhicules de moins d'un an, le coût de location est le prix d'achat du véhicule / 200, et pour les véhicules plus anciens, le coût de location est le prix d'achat / 250.
6. Écrire une méthode *peutTransporterPassagers(...)* qui à partir d'un nombre de passagers et d'un volume moyen par passager retourne vrai ou faux selon que l'autocar peut ou non transporter ces passagers.
7. Écrire la méthode infoCamion*()* qui retourne une chaîne de caractères (*String)* caractérisant un camion.

[1]La classes *Date* ou *Calendar* existent en Java, mais le but du TP n'est pas de les utiliser. Nous représenterons une date simplement par son année.
[2]Cette méthode est à implémenter dans la classe *Camion*.

8. Écrire la méthode infoAutocar*()* qui retourne une chaîne de caractères (*String)* caractérisant un autocar.

3. Classe contenant le programme principal

1. Écrire/complétez un programme de test créant une (instance de la classe) voiture Twingo, achetée hier 10000€, immatriculé 1234 AZ 49, avec un autoradio et nécessitant un permis B. Puis affichez sur la sortie standard[3] (l'écran) la chaîne de caractères retournée par la méthode *infoVoiture()*. Le résultat est-il celui escompté ? Si oui, passez à la question suivante.
2. Complétez ce programme de test pour créer un camion J9, acheté 20000€ il y a 5 ans, immatriculé 987 BCD 75, utilisable avec un permis B, et pouvant transporter 25 m^3. Affichez sur la sortie standard la chaîne de caractères retournée par la méthode *infoCamion()*. Affichez si ce camion peut transporter 7 m^3, on utilisera pour cela un appel à la méthode *peutTransporterVolume(...)*.
3. Afficher le coût de location de la Twingo précédemment créée, ainsi que celui du camion de type J9.
4. Créer dans le programme test un autocar de type FRI, acheté 90000€ en 2005, immatriculé 4567WX01, nécessitant un permis D, permettant de transporter 53 passagers et disposant d'une soute à bagages de 3 m^3. Affichez si cet autocar peut transporter 40 passagers ayant chacun 0.1 m^3 de bagages.
5. Les classes de l'application sont appelées à être complétées et utilisées par plusieurs programmeurs. Quelle solution proposez-vous, en ce qui concerne les champs (attributs et méthodes), pour assurer cette gestion.

[3] Utilisez la méthode *System.out.println(...)*. Respectez les minuscules/majuscules.

Quelques rappels de Java

Classes et attributs

Une classe se définit comme suit :

```
class NomClasse {
     // Attributs
     TypeAttribut nomAttribut;
     ...

     // Méthodes de la classe
     ...
}
```

Pour définir l'héritage d'une classe parent *C* par une classe *SC*, on définit *SC* de la façon suivante :

```
class SC extends C {
     // définition de la classe...
}
```

Une classe dont le nom est *NomClasse* doit obligatoirement être saisie dans un fichier de nom *NomClasse.java*. Pour compiler un tel fichier source Java, appeler le compilateur de la façon suivante :

```
javac NomClasse.java
```

Une fois la compilation exécutée, s'il n'y a pas d'erreur (auquel cas, les corriger), le compilateur a créé un fichier *NomClasse.class* qui contient la version compilée de la classe.

Types primitifs

Un entier se note *int,* un entier long *long, un réel float ou double*, et un booléen se note *boolean* ayant pour valeur *true* ou *false.*

Une chaîne de caractères est représentée par une instance de la classe *String...* Il ne s'agit donc pas d'un type primitif !

Méthodes

Les méthodes sont déclarées et définies à l'intérieur de la classe. La syntaxe de Java est très largement inspirée de celle de C / C++ :

- Le type de retour d'une méthode est déclaré juste avant son nom.
- Pour définir une méthode qui ne retourne pas de résultat, noter *void* suivi de la déclaration de la méthode.
- Les paramètres sont signalés par leur type puis leur nom, et séparés d'une virgule.
- L'affectation se fait par = et le test d'égalité par ==.

```
class Exemple
{
      void methode1() {
            System.out.println(”essai”);
      }
      int somme(int a, int b) {
            return a+b;
      }
      boolean positifP(int a) {
            if (a > 0)
                  return true;
            else
                  return false;
      }
}
```

Instanciation

Pour instancier une classe, c'est à dire créer un objet de cette classe, on doit déclarer une référence puis créer l'instance en utilisant *new* suivi d'un constructeur de la classe à instancier. Notez que la déclaration et la création de l'objet peut se faire sur une seule ligne.

```
NomClasse o;                // déclaration[4]
o = new NomClasse();        // création de l'objet
AutreClasse o2 = new AutreClasse(...);
```

Il est évidemment possible d'utiliser une référence comme type d'un attribut d'une classe, comme type d'un paramètre de méthode ou comme type de retour d'une méthode.

```
class Exemple {
   AutreClasse objetMemorise;

   AutreClasse obtenirValeur() {
       return objetMemorise;
   }
   void fixerValeur(AutreClasse i) {
       objetMemorise = i;
   }
}
```

Programme principal

Pour écrire un programme de test, il est nécessaire d'écrire une méthode particulière dans une nouvelle classe ou une des classes du projet. Cette méthode devra avoir la signature suivante :

```
class NomClasse {
   public static void main(String arg[]) {
       // code...
   }
}
```

Une classe possédant une telle méthode "main" peut être « exécutée ». C'est à dire qu'il est possible de lancer l'exécution de cette méthode main en appelant la machine virtuelle Java de la façon suivante (dans le répertoire contenant le fichier NomClasse.class obtenu après compilation) :

```
java NomClasse
```

Commentaires

```
// ceci est un commentaire sur une seule ligne
/* cela est
   un commentaire sur
   plusieurs lignes*/
```

Opérateurs arithmétiques

- opérateurs de bases : `+`, `-`, `*`, `/`
- opérateurs d'incrémentation et de décrémentation : `++` et `--`

Structure de contrôle conditionnel

```
if (condition) {
    instructions
}

if (condition) {
    instructions1
}
else {
    instructions2
}
```

Boucles

```
while (condition) {
    corps de la boucle
}

for (init ; condition ; progression) {
    corps
}
```

[4]Quand une référence est déclarée, elle est (implicitement) initialisée à la valeur *null*.