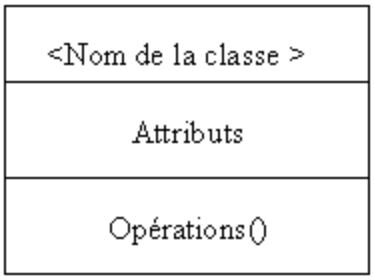
Bloc 2 - Ch. 7 – Les classes

1 Notion de classe

1.1 Notion de classe

Une classe va rassembler les objets de même nature, qui ont les mêmes attributs et le même comportement (méthodes ou opérations).

Représentation d'une classe :



Une classe avec UML (Unified Modeling Language) est représentée par un rectangle, sa partie haute est réservée à son nom, la zone du milieu contient les attributs et la partie basse les opérations.

Exemple :

Personne
prénom : String nom : String âge : int
getNom() : String sePrésenter() : void toString() : String manger() : void dormir() : void marcher() : void

Exercice 1 : Un train est caractérisé par sa vitesse et son état "enMarche" (qui est vrai ou faux). Un train peut démarrer, accélérer, ralentir, stopper.
Travail à faire : dessiner le diagramme de classe de la classe Train.

1.2 Notion de paquetage (package)

Un **paquetage**, ou ***package***, permet de regrouper des classes qui ont des fonctionnalités similaires ou qui appartiennent au même domaine métier.

Activité 1 : représenter le ***package*** `sncf` comprenant les classes qui nous avons vues (Train, Personne)

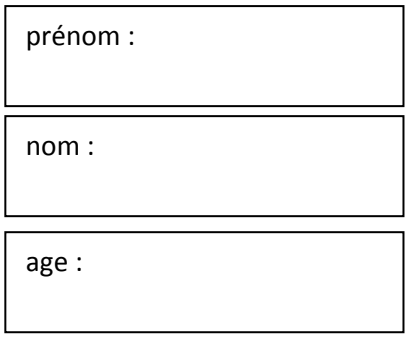
1.3 Ecriture de la classe Personne en Java

```
  public class Personne
  {
      String prenom;
      String nom;
      int age;

      public Personne()
      {
          this.nom = "";
          this.prenom = "";
          this.age = 0;
      }

      public void sePresenter()
      {
          System.out.println("Bonjour je suis " + prenom + " " +
nom);
      }

      public String getNom()
      {
          return nom;
      }
}
```



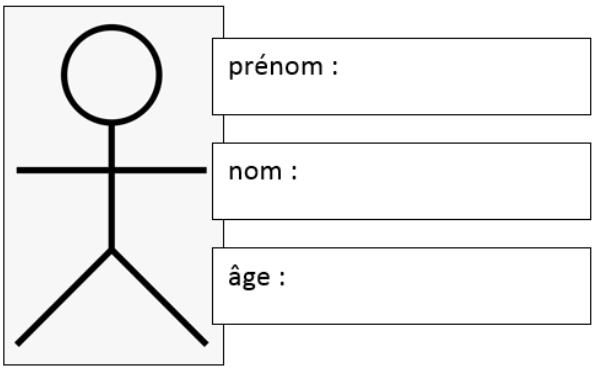

```
public class Program
{
  public static void main(String[] args)
  {
    // déclaration de l'objet
    Personne unePersonne;

    // INSTANCIATION de l'objet
    unePersonne = new Personne();

    // on VALORISE les attributs
    unePersonne.prenom = "Quentin";
    unePersonne.nom = "Tarantino";
    unePersonne.age = 58;

    unePersonne.sePresenter();

    String sonNom = unePersonne.getNom() ;
    System.out.println(sonNom) ;
  }
}
```



unePersonne

1.4 Ecriture de la classe Train en Java

```
package sncf ;                    // le paquetage de la classe

class Train
{
    // attributs
    int vitesse = 0 ;             // . . .
    boolean enMarche = false ;    // . . .

    // méthodes (ou fonctions)
    void demarrer()               // . . .
    {
        System.out.println("Démarrage");    //  . . .
    }

    void stopper()                // . . .
    {
        System.out.println("Arrêt");
    }
}

class Program
{
    // Méthode main : programme principal
    public static void main(String[] args)
    {
        // ici, le programme principal
    }
}
```

Exercice 2 : Tracez la vitesse et l'état du train lors de l'exécution du programme

```
public static void main(String[] args)      // . . .
{
   Train unTrain;                 // . . .

   unTrain = new Train() ;        // . . .


   unTrain.enMarche = true ;

   unTrain.demarrer();

   unTrain.vitesse = 15;

   unTrain.vitesse = 30;

   unTrain.vitesse = 0;

   unTrain.stopper();

   unTrain.enMarche = false;

}
```

unTrain

vitesse		enMarche	

vitesse		enMarche	

vitesse		enMarche	

vitesse		enMarche	

vitesse		enMarche	

vitesse		enMarche	

vitesse		enMarche	

2 L'encapsulation

2.1 Visibilité des attributs et opérations

Les attributs et méthodes sont associés à des indicateurs de visibilité. Ces indicateurs sont très utiles pour cacher à l'utilisateur la représentation du type de l'objet ainsi que les procédures et fonctions non concernées par les spécifications. Généralement on compte trois niveaux de visibilité - du plus restrictif au plus accessible appelés **modificateurs d'accès** :

Privé (**private**) : rend l'élément visible à la classe seule.

Protégé (**protected**) : rend l'élément visible aux sous-classes de la classe (la notion de sous-classe sera abordée avec l'héritage)

Public : rend l'élément visible à tout client de la classe.

Le niveau de visibilité est symbolisé par les caractères +, # et -, qui correspondent respectivement aux niveaux public, protégé, et privé (notation utilisée par la méthode UML). La nouvelle représentation devient :

Train
- vitesse - enMarche
+ accélérer() + démarrer() + ralentir() + stopper()

2.2 L'encapsulation

L'**encapsulation** représente un des concepts fondamentaux du monde objet. Par principe même, on ne peut accéder aux attributs ou opérations d'un objet en dehors de ceux explicitement déclarés publics par la classe. Ainsi une classe n'est utilisable qu'à travers son interface - ensemble des attributs et opérations publics.

Une recommandation forte impose de ne pas déclarer les attributs comme publics.

Activité 2 : Encapsulation.
Travail à faire : 1) Commenter le code. 2) Repérer les erreurs.

```
class Train
{
    // attributs . . .
    private int vitesse;            // . . .
    private boolean enMarche;       // . . .

    // méthodes . . .
    public void demarrer()         {…}
    public void stopper()          {…}

    public int getVitesse()         // . . .
    {
        return vitesse ;            // . . .
    }

    public void setVitesse(int uneVitesse)     // . . .
    {
        vitesse = uneVitesse ;                 // . . .
    }
}
```

```
// On déplace la méthode main dans une classé séparée nommée Program :
class Program
{
    public static void main(String[] args)      // . . .
    {
        Train t = new Train();      // . . .

        t.vitesse = 10;             // . . .

        t.setVitesse(10);           // . . .

        System.out.println( t.vitesse ) ;   // . . .

        int v = t.getVitesse();     // . . .
        System.out.println( v ) ;   // . . .
    }
}
```

La donnée **vitesse** est encapsulée dans la classe **Train**, elle n'est pas accessible directement.

Remarque : les méthodes présentées ci-dessus sont publiques, mais on peut aussi définir des méthodes privées.

3 Les méthodes des classes en Java

Le rôle d'une méthode est d'effectuer un traitement en utilisant, en général, les données (c. à d. les attributs). Les méthodes peuvent être publiques ou privées. L'ensemble des méthodes publiques représente l'interface de la classe, sa partie "utilisable" à l'extérieur. On peut les classer en trois groupes selon leurs fonctionnalités.

3.1 Les méthodes de création : constructeurs

Java propose des méthodes particulières qui ont comme rôle d'initialiser les attributs, ce sont les constructeurs.
Un **constructeur** est appelé automatiquement au moment de la création de l'objet à l'aide de l'opérateur **new**.

Les constructeurs ont une signature particulière :

- Ils portent le nom de la classe.
- Ils ne retournent rien, pas même **void**.

Exemple:

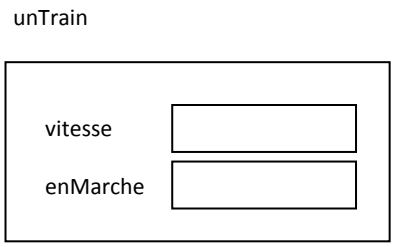
```
public Train(int uneVitesse, boolean unEtat)
{
    this.vitesse = uneVitesse;
    this.enMarche = unEtat;
}
```

Chaque objet de la classe Train sera instancié en utilisant le constructeur :

```
Train p = new Train(20, true);
```

Exercice 3 – Complétez le schéma à droite (représentation en mémoire de l'objet **unTrain**)

```
public static void main(String[] args)
{
    Train unTrain = new Train(20, true);
}
```



On peut écrire plusieurs constructeurs pour une classe :

```
    // constructeur sans paramètre
    public Train()
    {
        vitesse = 0;
        enMarche = false;
    }
```

Le mécanisme permettant de définir plusieurs méthodes ayant le même nom (mais pas la même signature) s'appelle une **surcharge**.

```
Train t1 = new Train();             // appel du constructeur sans paramètre
Train t2 = new Train (50, true);    //appel du constructeur à deux arguments
```

Le langage appellera le constructeur correspondant aux arguments passés au moment de l'appel.

Si aucun constructeur n'est défini explicitement, Java va générer automatiquement un constructeur par défaut qui va initialiser chaque attribut selon son type. Il est fortement conseillé de toujours fournir au moins un constructeur à chaque classe.

3.2 Les méthodes : accesseurs/modificateurs

3.2.1 Les accesseurs en lecture

Ils permettent de donner l'état de l'objet sans le modifier :

```
public class Train
{
    // attributs . . .
    private int vitesse;
    private boolean enMarche;

    //  . . .

    public int getVitesse()
    {
        return this.vitesse;
    }

}
```

Remarque : nous avons utilisé le mot réservé **this** afin de faire référence à l'objet courant, cette écriture (non indispensable) améliore la lisibilité.

Exercice 4 – Ecrivez le code de l'accesseur **isEnMarche** (on utilise **is** au lieu de **get** car **enMarche** est un booléen)

3.2.2 Les modificateurs

Ils permettent de modifier l'état de l'objet :

```
// . . .
// on utilise en général le préfixe set.
    public void setVitesse(int v)
    {
        this.vitesse = v;
    }
```

Exercice 5 – Ecrivez le code de l'accesseur **setEnMarche**

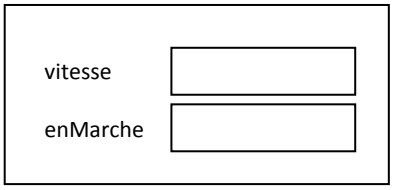
Exercice 6 – complétez le schéma

```
  public static void main(String[] args)
  {
      Train unTrain = new Train(50, true);

      unTrain.setVitesse ( 100 );

  }
```

unTrain



3.3 Les méthodes agissant sur le comportement ou l'état de l'objet.

Ce sont toutes les autres méthodes. Exemple :

```
    public void stopper()           // .  .  .
    {
        vitesse = 0;
        enMarche = false;
    }

    public void ralentir(int v)     // .  .  .
    {
        vitesse = vitesse - v;
    }

    public void accelerer(int v)    // .  .  .
    {
        enMarche = true;
        vitesse = vitesse + v;
    }
```

TP 10 – Classe Train

TP 11 – Haras de Lamballe