**Haute École Bruxelles-Brabant**
**École Supérieure d'Informatique**
Rue Royale, 67. 1000 Bruxelles
02/219.15.46 – esi@he2b.be

# Algorithmique

2020

Bachelor en Informatique
DEV2

A. Paquot (APA), M. Codutti (MCD), N. Richard (NRI),
S. Drobisz (SDR), S. Rexhep (SRE) & ???

Document produit avec LaTeX.
Version du 14 janvier 2020.

# Table des matières

Chapitre 1

# L'orienté objet

Le cours de Java vous a présenté la programmation orienté objet. Dans ce chapitre, nous allons rapidement revoir ce sujet et présenter comment nous allons l'utiliser dans ce cours. Nous nous contenterons de parler d'*encapsulation*. Les autres piliers de l'orienté objet (*héritage* et *polymorphisme*) ne seront pas vus cette année.

## 1.1 Motivation

Au cours de Java, vous avez vu que l'orienté objet permet de structurer une application en regroupant dans un même *objet* des données et le code qui va manipuler ces données.

Une autre façon de voir l'orienté objet est de constater qu'une classe permet de définir un nouveau *type de données*. On pourra préciser les valeurs possibles et ce qu'on peut en faire. C'est ainsi que nous l'utiliserons pour définir les listes dans un prochain chapitre.

## 1.2 Illustration : une durée

Voyons tout cela au travers d'un exemple complet. Il est parfois utile d'avoir à sa disposition un type de données permettant de représenter une durée. Utiliser plusieurs entiers (un pour les heures, un autre pour les minutes, un autre encore pour les secondes) n'est pas pratique. Voyons comment définir ce nouveau type de données en orienté objet.

### 1.2.1 Ce que l'on veut vraiment

Avant tout, il faut bien préciser ce que l'on veut décrire et bien faire la distinction entre un *moment* et une *durée*. L'« heure » est un concept multifacette. Parle-t-on de l'heure comme moment dans la journée ou de l'heure comme représentant une durée ? Dans le premier cas, elle ne peut dépasser 24h et la différence entre 2 heures n'a pas de sens (ou plus précisément n'est pas une heure, mais une durée !). Ce que nous nous proposons de créer ici est une durée, correspondant au deuxième cas. Et pour être plus précis encore, nous allons nous limiter à une précision à la seconde près, pas plus [1].

1. Ajouter plus de précision ne serait pas plus compliqué à faire.

### 1.2.2 Le comportement (les méthodes)

La première question à se poser est celle des services que l'on veut fournir, c'est-à-dire des méthodes publiques de la classe. On doit pouvoir *construire* une durée. On doit pouvoir connaitre le nombre de jours, d'heures, minutes ou secondes correspondant à une durée. On doit pouvoir effectuer des calculs avec des durées (addition, soustraction). Enfin, on doit pouvoir comparer des durées. Arrêtons-nous là, mais en pratique, on pourrait trouver encore bon nombre d'autres méthodes qu'il serait intéressant de fournir.

Voici comment nous allons noter tout cela au cours d'algorithmique.

```
class Durée
    public :
        constructor Durée(secondes : entier)
        constructor Durée(heure, minute, seconde : entiers)

        method getJour() → entier                          // nb de jours dans une durée

        method getHeure() → entier                         // entier entre 0 et 23 inclus
        method getMinute() → entier                        // entier entre 0 et 59 inclus
        method getSeconde() → entier                       // entier entre 0 et 59 inclus

        method getTotalHeures() → entier                   // Le nombre total d'heures
        method getTotalMinutes() → entier                  // Le nombre total de minutes
        method getTotalSecondes() → entier                 // Le nombre total de secondes

        method ajouter(autreDurée : Durée)
        method différence(autreDurée : Durée) → Durée
        method plusPetit(autreDurée : Durée) → booléen
end
```

⚠ **Quelques remarques**

- On a deux constructeurs, ce qui offre plus de souplesse pour initialiser un objet. On parle de « **surcharge** » des constructeurs.
- Faisons bien la distinction entre les méthodes getXXX() et getTotalXXX(). Par exemple, la méthode getMinute() retourne la valeur de la composante « minutes » dans une représentation HMS tandis que la méthode getTotalMinutes() retourne le nombre total de minutes entières pour cette durée. Ex : pour 1h23'12", getMinute() retourne 23 et getTotalMinutes() retourne 83. Idem avec les heures et les secondes.
- Les méthodes getTotalXXX() retournent le nombre (toujours entier) de XXX contenus dans la durée. Exemple, avec la durée 0h23'52", getTotalMinutes() retourne 23 et pas 24 (autrement dit, il n'y a pas d'arrondi vers le haut).
- Il n'y a pas de *mutateur* (setXXX()). Ce qui signifie qu'on ne peut pas changer directement la valeur de l'objet après son initialisation. Les seules modifications viendront de la méthode ajouter(). On aurait pu définir des mutateurs mais nous n'avons pas jugé utile de le faire dans ce cas précis. Vous verrez dans le cours de Java des motivations à ce choix.
- La méthode ajouter() ne retourne rien. En effet, elle ajoute la durée à l'objet sur lequel est appelée la méthode. C'est un choix ; on aurait aussi pu dire que la méthode ne modifie pas l'objet mais en retourne un autre qui représente la somme. Dans ce cas, on l'aurait plutôt appelée « plus( ) ».
- La méthode différence(), elle, renvoie toujours une durée (positive).
- Nous ne définissons pas de méthode d'affichage similaire au toString() qu'on retrouve en Java. L'affichage correct de l'information ne fait pas partie des préoccupations de ce cours. On supposera que "**afficher** objet" affiche correctement les données associées à l'objet.

### 1.2.3 La représentation de l'état (les attributs)

La question suivante est : « Comment représenter une durée en interne ? ». Plusieurs possibilités existent. Par exemple :

- via le nombre d'heures, de minutes et de secondes
- via le nombre total de secondes
- via une chaine, par exemple au format « HH : MM : SS » où HH pourrait éventuellement excéder 23.

Le premier choix semble le plus évident mais réfléchissons-y de plus près. D'une part, pourquoi se limiter aux heures. On pourrait introduire un champ 'jour' (après tout on a bien une méthode getJour()). Quel critère doit vraiment nous permettre de décider ? Il faut une représentation qui soit suffisante (tout est représenté) et qui permette d'écrire des méthodes lisibles et si possible efficaces (c'est-à-dire où le calcul est rapide). Selon ces critères, la deuxième représentation est de loin la meilleure.

Voilà comment nous indiquons les attributs d'une classe.

```
class Durée
    private :
        totalSecondes: entier
    public :
        // Ici viennent les constructeurs et les méthodes
end
```

Pour rappel de votre cours de langage, ce qui est privé n'est pas utilisable directement par du code extérieur à la classe. Un code extérieur, manipulant des objets de cette classe, ne peut utiliser que ce qui est public.

### 1.2.4 L'implémentation

On est à présent prêt pour écrire le code des méthodes. Pour une meilleure lisibilité, nous gardons les signatures des méthodes dans la classe et nous détaillons leur contenu en dehors. Ce qui donne :

```
class Durée
    private :
        totalSecondes: entier
    public :
        constructor Durée(secondes : entier)
        constructor Durée(heure, minute, seconde : entiers)

        method getJour() → entier                          // nb de jours dans une durée
        method getHeure() → entier                         // entier entre 0 et 23 inclus
        method getMinute() → entier                        // entier entre 0 et 59 inclus
        method getSeconde() → entier                       // entier entre 0 et 59 inclus

        method getTotalHeures() → entier                   // Le nombre total d'heures
        method getTotalMinutes() → entier                  // Le nombre total de minutes
        method getTotalSecondes() → entier                 // Le nombre total de secondes

        method ajouter(autreDurée : Durée)
        method différence(autreDurée : Durée) → Durée
        method plusPetit(autreDurée : Durée) → booléen
end
```

```
constructor Durée(secondes : entier)
    if secondes < 0
        error "paramètre négatif" [a]
    end
    totalSecondes = secondes
end

constructor Durée(heure, minute, seconde : entiers)
    if heure < 0 OU minute < 0 OU seconde < 0 OU minute>59 ou seconde>59
        error "un des paramètres est invalide"
    end
    totalSecondes = 3600*heure + 60*minute + seconde
end

// Retourne le nombre de jours dans une représentation JJ/HH :MM :SS
method getJour() → entier
    return totalSecondes DIV (3600*24)
end

// Retourne le nombre d'heures dans une représentation JJ/HH :MM :SS
method getHeure() → entier
    return (totalSecondes DIV 3600) MOD 24                    // On doit enlever les jours éventuels
end

// Retourne le nombre de minutes dans une représentation JJ/HH :MM :SS
method getMinute() → entier
    return (totalSecondes DIV 60) MOD 60                    // On doit enlever les heures éventuelles
end

// Retourne le nombre de secondes dans une représentation JJ/HH :MM :SS
method getSeconde() → entier
    return totalSecondes MOD 60                            // On doit enlever les minutes éventuelles
end

// Retourne le nombre entier d'heures complètes
method getTotalHeures() → entier
    return totalSecondes DIV 3600
end

// Retourne le nombre entier de minutes complètes
method getTotalMinutes() → entier
    return totalSecondes DIV 60
end

// Retourne le nombre entier de secondes complètes
method getTotalSecondes() → entier
    return totalSecondes
end

method ajouter(autreDurée : Durée)
    totalSecondes = totalSecondes + autreDurée.getTotalSecondes()
end

method différence(autreDurée : Durée) → Durée
    return new Durée(valeurAbsolue(totalSecondes - autreDurée.getTotalSecondes()))
end

method plusPetit(autreDurée : Durée) → booléen
    return totalSecondes < autreDurée.getTotalSecondes()
end
```

*a.* L'instruction **error** indique que l'algorithme ne peut pas poursuivre normalement. Il s'arrête avec un message d'erreur.

### 1.2.5 Utilisation

Pour utiliser le nouveau type de donnée créé, il faut l'instancier, c'est-à-dire créer un nouvel objet de ce type. Nous allons utiliser le mot clé **new** pour rester très proche de Java.

Illustrons cela au travers d'un petit algorithme qui calcule la différence entre deux durées.

```
algorithm diffDurée
    durée1, durée2: Durée                    // Les variables sont déclarées
    durée1 = new Durée(3, 4, 49)             // durée1 est créé
    durée2 = new Durée(3, 24, 37)            // durée2 est créé
    print durée2.différence(durée1)
end
```

### 1.2.6 Version Java

Voici ce que ça pourrait donner en Java :

```java
package dev2.algo.oo;

import java.util.Objects;

/**
 * Représente une durée avec une précision à la seconde.
 */
public class Duree {

   private int totalSecondes;

   /**
    * Construit une durée donnée par un total de secondes.
    *
    * @param secondes le nombre total de secondes de la durée.
    * @throws IllegalArgumentException si <code>secondes</code> est négatif.
    */
   public Duree(int secondes) {
      if (secondes < 0) {
         throw new IllegalArgumentException("Nb de secondes négatif: " + secondes);
      }
      totalSecondes = secondes;
   }

   /**
    * Construit une durée donnée par des heures/minutes/secondes.
    *
    * @param heure le nombre d'heures de la durée.
    * @param minute le nombre de minutes de la durée.
    * @param seconde le nombre de secondes de la durée.
    * @throws IllegalArgumentException si :
    * heure est négatif ou minute n'est pas entre 0 et 59
    * ou seconde n'est pas entre 0 et 59
    */
   public Duree(int heure, int minute, int seconde) {
      if (heure < 0) {
         throw new IllegalArgumentException("heure invalide: " + heure);
      }
      if (minute < 0 || minute > 59) {
         throw new IllegalArgumentException("minute invalide: " + minute);
      }
      if (seconde < 0 || seconde > 59) {
         throw new IllegalArgumentException("seconde invalide: " + seconde);
      }
      totalSecondes = 3600 * heure + 60 * minute + seconde;
   }
```

```java
    /**
     * Donne le nombre de jours dans une représentation JJ/HH:MM:SS.
     *
     * @return le nombre de jours dans une représentation JJ/HH:MM:SS.
     */
    public int getJour() {
        return totalSecondes / (24 * 3600);
    }

    /**
     * Donne le nombre d'heures dans une représentation JJ/HH:MM:SS.
     *
     * @return le nombre d'heures dans une représentation JJ/HH:MM:SS.
     */
    public int getHeure() {
        return (totalSecondes / 3600) % 24;
    }

    /**
     * Donne le nombre de minutes dans une représentation JJ/HH:MM:SS.
     *
     * @return le nombre de minutes dans une représentation JJ/HH:MM:SS.
     */
    public int getMinute() {
        return (totalSecondes / 60) % 60;
    }

    /**
     * Donne le nombre de secondes dans une représentation JJ/HH:MM:SS.
     *
     * @return le nombre de secondes dans une représentation JJ/HH:MM:SS.
     */
    public int getSeconde() {
        return totalSecondes % 60;
    }

    /**
     * Donne le nombre total d'heures entières de la durée.
     *
     * @return le nombre total d'heures entières de la durée.
     */
    public int getTotalHeures() {
        return totalSecondes / 3600;
    }

    /**
     * Donne le nombre total de minutes entières de la durée.
     *
     * @return le nombre total de minutes entières de la durée.
     */
    public int getTotalMinutes() {
        return totalSecondes / 60;
    }

    /**
     * Donne le nombre total de secondes de la durée.
     *
     * @return le nombre total de secondes de la durée.
     */
    public int getTotalSecondes() {
        return totalSecondes;
    }

    /**
     * Ajoute une durée. La durée donnée en paramètre est ajoutée à cette durée
     * qui est ainsi modifiée.
     *
     * @param autre la durée à ajouter.
     */
    public void ajouter(Duree autre) {
        totalSecondes += autre.totalSecondes;
    }
```

```
121     /**
122      * Différence entre deux durées. Calcule une nouvelle durée représentant la
123      * différence entre cette durée et l'autre durée donnée en paramètre.
124      *
125      * @param autre une seconde durée.
126      * @return la dférence entre les durées.
127      */
128     public Duree différence(Duree autre) {
129         return new Duree(totalSecondes - autre.totalSecondes);
130     }
131 
132     /**
133      * Teste si cette durée est plus petite qu'une autre donnée en paramètre.
134      *
135      * @param autre une seconde durée.
136      * @return <code>true</code> si cette durée est inférieure à celle donnée en
137      * paramètre.
138      */
139     public boolean plusPetit(Duree autre) {
140         return totalSecondes < autre.totalSecondes;
141     }
142 
143     @Override
144     public int hashCode() {
145         return Objects.hash(totalSecondes);
146     }
147 
148     @Override
149     public boolean equals(Object obj) {
150         if (this == obj) {
151             return true;
152         }
153         if (obj == null) {
154             return false;
155         }
156         if (getClass() != obj.getClass()) {
157             return false;
158         }
159         final Duree other = (Duree) obj;
160         return (this.totalSecondes == other.totalSecondes);
161     }
162 
163     /**
164      * Représentation textuelle de la durée.
165      * @return la durée sous la forme "JJ/HH:MM:SS"
166      */
167     @Override
168     public String toString() {
169         String res;
170         if (getJour() == 0) {
171             res = "";
172         } else {
173             res = (getJour() + "/");
174         }
175         res += String.format("%02d:%02d:%02d", getHeure(), getMinute(), getSeconde());
176         return res;
177     }
178 
179     public static void main(String[] args) {
180         Duree durée1;
181         Duree durée2;
182         durée1 = new Duree(3, 4, 49);
183         durée2 = new Duree(3, 24, 37);
184         System.out.println("Durée 1 = " + durée1);
185         System.out.println("Durée 2 = " + durée2);
186         System.out.println("Différence = " + durée2.différence(durée1));
187         Duree gdDurée = new Duree(123456789);
188         System.out.println("Gd Durée = " + gdDurée);
189     }
190 }
```

## 1.3 Quelques éléments de syntaxe

Clarifions certaines notations liées aux objets.

- ▷ Pour un attribut brol, on choisira de nommer l'accesseur [2] getBrol et le mutateur [3] setBrol. Dans le cas particulier d'un attribut booléen, on pourra appeler l'accesseur isBrol ou encore estBrol.
- ▷ On peut directement afficher un objet. Cela affiche l'état d'un objet d'une façon claire pour l'utilisateur [4].

```
rendezVous: Durée
rendezVous = new Durée(14, 23, 56)
print rendezVous                    // affichera 14, 23 et 56 dans un format lisible.
```

- ▷ De même, on peut directement lire un objet, ce qui a pour effet de créer un objet avec un état correspondant aux valeurs lues pour ses attributs.

```
rendezVous: Durée
read rendezVous
```

- ▷ La comparaison de deux objets est toujours un problème délicat en orienté objet. Nous nous baserons ici sur les conventions Java et utiliserons la notation o1.égale(o2) quand il s'agira de vérifier que o1 et o2 sont dans le même état, c'est-à-dire que leurs attributs ont la même valeur.
- ▷ Lorsqu'on déclare un objet, il n'est pas encore créé. On peut utiliser la valeur spéciale « rien » pour indiquer ou tester qu'un objet n'est pas encore créé.

```
parcours: Durée                     // parcours = rien
parcours = new Durée( 14, 23, 56 )  // parcours ≠ rien
if parcours ≠ rien
|   parcours = rien                // parcours = rien
end
```

- ▷ Si une classe ne propose pas de constructeur, on peut néanmoins instancier un objet (via new NomClasse()). On considère dans ce cas que les attributs ne sont pas initialisés.

2. Pour rappel, un *accesseur* est une méthode donnant la valeur d'un attribut.
3. Pour rappel, un *mutateur* est une méthode permettant de modifier la valeur d'un attribut.
4. Le format précis n'est pas spécifié car il n'est pas important pour ce cours.

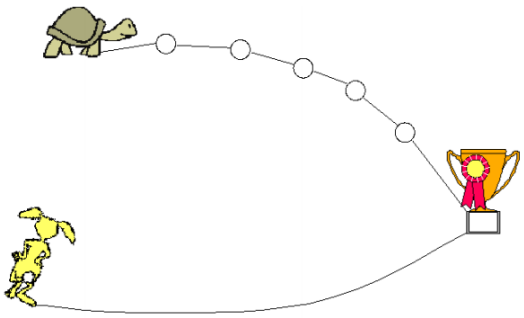
## 1.4 Mise en pratique : le lièvre et la tortue

Partons d'un petit jeu, « Le lièvre et la tortue » [5], et voyons comment le coder en OO.

### 1.4.1 Description du jeu

Un lièvre et une tortue font une course. Le lièvre est plus rapide que la tortue. Pour donner plus de chance à la tortue de gagner une course de 6 km, on adopte la règle de jeu suivante :

- ▷ On lance un dé.
- ▷ Si le 6 sort, le lièvre est autorisé à démarrer et gagne la course ; sinon on laisse la tortue avancer d'un kilomètre.
- ▷ On recommence le procédé jusqu'à la victoire du lièvre ou de la tortue.



### 1.4.2 Solution non orientée objet

Pour ne pas aller trop vite et vous perdre tout de suite, commençons par une version non orientée objet du jeu.

#### Représenter le jeu

Il faut d'abord se poser cette question : Comment représenter le jeu ? Une représentation du jeu doit être complète. C'est-à-dire qu'à partir de cette représentation, on doit pouvoir indiquer exactement où on en est dans le jeu et pouvoir le poursuivre.

Pour le dire autrement, imaginons qu'on joue à ce jeu « en vrai », sur une table de jeu. La représentation informatique doit capturer tout ce qui est pertinent dans le jeu physique de sorte que, si on range la boite de jeu, on peut, le lendemain, reconstruire le jeu exactement comme il était.

Dans notre exemple, cela veut dire quoi ?

**La tortue**
: On doit pouvoir savoir où elle en est dans son avancée. Un petit entier (de 0 à 6) reprenant son avancée en km suffit. Appelons-le avancéeTortue par exemple.

**Le lièvre**
: Pendant le jeu, il est en permanence au départ et lorsque sort un 6, il atteint directement l'arrivée. Plusieurs possibilités s'offrent à nous :

1. On pourrait imaginer un entier valant 0 ou 6, appelé avancéeLièvre.
2. On pourrait aussi imaginer un booléen à vrai lorsqu'il est au départ et faux lorsqu'il est à l'arrivée. On pourrait l'appeler lièvreAuDépart.

5. Lu sur le net : http://mathemathieu.free.fr/2b/doc/pb_algo/problemes_et_algorithmique.pdf (site désormais inaccessible)

3. Ou encore un booléen ayant exactement le sens inverse. Le nom devra être choisi judicieusement pour ne pas induire le lecteur en erreur. Par exemple, ici, on pourrait choisir lièvreArrivé.

Ici, le troisième choix nous semble le plus pertinent.

**Le dé**
Un entier de 1 à 6 suffit pour représenter le résultat d'un dé. Appelons-le simplement dé.

**Les joueurs**
Si on observe un jeu physique, on peut s'attarder sur les personnes en train de jouer. Faut-il les représenter ? On pourrait imaginer de connaitre leur nom, le nombre de fois qu'elles ont joué à ce jeu, leur nombre de victoires. Dans l'énoncé, rien n'indique qu'il faille tenir compte de tout cela. On s'intéresse au jeu proprement dit, et c'est tout.

**Le plateau**
On peut imaginer que dans le jeu physique, il y aurait une sorte de plateau avec des km indiqués sur lequel avancerait la tortue. Mais il n'y a aucune information changeante sur ce plateau qui vaille la peine d'être retenue.

### Un macro algorithme

Avant de se lancer dans l'écriture d'une solution détaillée du jeu, commençons par une solution non détaillée et voyons si tout semble clair et faisable.

```
algorithm jeuLièvreTortue
    Initialiser le jeu
    while le jeu n'est pas fini
        Lancer le dé
        if le dé est 6
            Le lièvre est arrivé
        else
            La tortue avance
        end
        print l'état du jeu
    end
    print le vainqueur
end
```

### Détailler l'algorithme

Repassons à présent sur l'algorithme et vérifions que nous pouvons détailler chacun des points restés généraux.

- **Initialiser le jeu.** Il suffit de placer la tortue en 0 et d'indiquer que le lièvre n'est pas encore arrivé. La valeur initiale du dé n'a pas d'importance.
- **Le jeu n'est pas fini.** Le jeu sera fini lorsque le lièvre sera arrivé (ce qu'on peut tester grâce au booléen lièvreArrivé) ou que la tortue sera au km 6.
- **Lancer le dé.** C'est trivial si on utilise l'algorithme hasard() à notre disposition.
- **Le lièvre est arrivé.** Il suffit de mettre le booléen lièvreArrivé à vrai.
- **La tortue avance.** C'est trivial.
- **Afficher l'état du jeu.** À savoir, sur quelle face est tombé le dé et où se trouvent à présent le lièvre et la tortue.
- **Afficher le vainqueur.** Ce sera le lièvre si son booléen est à vrai et la tortue sinon (dans ce cas, son avancée sera forcément de 6 puisque le jeu est fini).

Au final, on obtient :

```
algorithm jeuLièvreTortue
    avancéeTortue: integer
    lièvreArrivé: boolean
    dé: integer

    avancéeTortue = 0
    lièvreArrivé = faux

    while avancéeTortue < 6 ET NON lièvreArrivé
        dé = hasard(6)
        if dé = 6
            lièvreArrivé = vrai
        else
            avancéeTortue++
        end
        print dé, avancéeTortue, lièvreArrivé
    end

    if lièvreArrivé
        print "Le lièvre a gagné"
    else
        print "La tortue a gagné"
    end
end
```

En Java :

```java
package dev2.algo.oo.lievretortue;

import mcd.util.Hasard;

/**
 * Version non OO du jeu de lièvre et de la tortue.
 */
public class VersionNonOO {

   public static void main(String[] args) {
      int avancéeTortue;
      boolean lièvreArrivé;
      int dé;

      avancéeTortue = 0;
      lièvreArrivé = false;
      while (avancéeTortue < 6 && !lièvreArrivé) {
         dé = Hasard.dice();     // Hasard est une classe utilitaire personnelle
         if (dé == 6) {
            lièvreArrivé = true;
         } else {
            avancéeTortue++;
         }
         afficherÉtat(dé, avancéeTortue, lièvreArrivé);
      }
      if (lièvreArrivé) {
         System.out.println("Le lièvre a gagné");
      } else {
         System.out.println("La tortue a gagné");
      }
   }

   public static void afficherÉtat(int dé, int avancéeTortue, boolean lièvreArrivé) {
      System.out.println("dé=" + dé + ", tortue=" + avancéeTortue + ", lièvre=" + (lièvreArrivé?6:0));
   }
}
```

### 1.4.3 Solution orientée objet

Voyons à présent ce que ça pourrait donner si on introduit de l'orienté objet. Examinons d'abord les objets physiques du jeu.

**La tortue**

On peut envisager de définir une classe pour la tortue. Une tortue a une avancée. Au départ, elle est au kilomètre 0. Elle peut avancer d'un kilomètre à la fois. Elle a fini et gagne lorsqu'elle arrive au kilomètre 6.

```
class Tortue
    private :
        avancée: integer
    public :
        constructor Tortue
            avancée = 0
        end

        method avancer
            avancée++
        end

        method estArrivée() → boolean
            return avancée = 6
        end

        method getAvancée() → integer
            return avancée
        end
end
```

Remarquez qu'on n'introduit pas de mutateur car on veut que la tortue n'avance qu'en respectant les règles du jeu.

**Le lièvre**

On peut appliquer la même démarche pour le lièvre qui aurait un attribut booléen indiquant s'il est arrivé ou pas.

**Le dé**

Le dé est également un objet de notre jeu et peut être défini via une classe.

```
class Lièvre
    private :
        arrivé: boolean
    public :
        constructor Lièvre
            arrivé = faux
        end

        method avancer
            arrivé = vrai
        end

        method estArrivé() → boolean
            return arrivé
        end
end
```

```
class Dé
    private :
        valeur: integer
    public :
        constructor Dé
        end

        method lancer
            valeur = hasard(6)
        end

        method getValeur() → integer
            return valeur
        end
end
```

### L'algorithme du jeu

L'algorithme du jeu peut être récrit en utilisant les trois classes qu'on vient de définir.

```
algorithm jeuLièvreTortue
    tortue: Tortue
    lièvre: Lièvre
    dé: Dé

    tortue = new Tortue()
    lièvre = new Lièvre()
    dé = new Dé()

    while NON tortue.estArrivée() ET NON lièvre.estArrivé()
        dé.lancer()
        if dé.getValeur()=6
            lièvre.avancer()
        else
            tortue.avancer()
        end
        print dé.getValeur(), tortue.getAvancée(), lièvre.estArrivé()
    end

    if lièvre.estArrivé()
        print "Le lièvre a gagné"
    else
        print "Le tortue a gagné"
    end
end
```

Cette solution OO est plus longue en terme de nombre de lignes mais elle est **plus lisible** car la méthode principale fait intervenir des termes qui sont proches du problème.

En Java :

```java
package dev2.algo.oo.lievretortue;

/**
 * Une tortue dans le jeu du lièvre et de la tortue.
 */
public class Tortue {

   private int avancée;

   public Tortue() {
      avancée = 0;
   }

   public void avancer() {
      avancée++;
   }

   public boolean estArrivée() {
      return avancée == 6;
   }

   public int getAvancée() {
      return avancée;
   }
}
```

```java
package dev2.algo.oo.lievretortue;

/**
 * Un lièvre dans le jeu du lièvre et de la tortue.
 */
public class Lievre {

   private boolean arrivé;

   public Lievre() {
      arrivé = false;
   }

   public void avancer() {
      arrivé = true;
   }

   public boolean estArrivé() {
      return arrivé;
   }
}
```

```java
package dev2.algo.oo.lievretortue;

import mcd.util.Hasard;

/**
 * Un dé à 6 faces.
 */
public class De {

   private int valeur;

   public void lancer() {
      valeur = Hasard.in(1, 6);
   }

   public int getValeur() {
      return valeur;
   }
}
```

```java
package dev2.algo.oo.lievretortue;

/**
 * Version OO du jeu du lièvre et de la tortue.
 */
public class VersionOO {

   public static void main(String[] args) {
      Tortue tortue;
      Lievre lièvre;
      De dé;

      tortue = new Tortue();
      lièvre = new Lievre();
      dé = new De();

      while (!tortue.estArrivée() && !lièvre.estArrivé()) {
         dé.lancer();
         if (dé.getValeur() == 6) {
            lièvre.avancer();
         } else {
            tortue.avancer();
         }
         afficherÉtat(dé.getValeur(), tortue.getAvancée(), lièvre.estArrivé());
      }
      if (lièvre.estArrivé()) {
         System.out.println("Le lièvre a gagné");
      } else {
         System.out.println("La tortue a gagné");
      }
   }

   public static void afficherÉtat(int dé, int avancéeTortue, boolean lièvreArrivé) {
      System.out.println("dé=" + dé + ", tortue=" + avancéeTortue + ", lièvre=" + (lièvreArrivé ? 6 : 0));
   }

}
```

Est-ce une bonne idée d'avoir défini ces trois classes ? C'est une question qu'il est légitime de se poser quand les classes sont aussi simples. Remarquons toutefois que le code est plus modulaire et que la méthode principale est plus facile à lire.

La classe Dé se justifie d'autant plus qu'elle pourra probablement servir à de nombreuses occasions. Ce sera encore plus le cas si on la généralise à des dés qui n'ont pas forcément 6 faces.

```
class Dé
    private :
        nbFaces: integer
        valeur: integer
    public :
        constructor Dé(nf: integer)
            nbFaces = nf
        end

        method lancer
            valeur = hasard(nbFaces)
        end

        method getValeur() → integer
            return valeur
        end
end
```

Dans l'algorithme principal, le seul changement est la création du dé qui devient :

```
dé = new Dé(6)
```

### 1.4.4 Solution MVC (« Modèle-Vue-Contrôleur »)

Dans la version OO qu'on vient de voir, on a introduit trois classes mais il reste tout un morceau, l'algorithme principal, qui n'est pas OO. Peut-on aller plus loin dans l'OO ? Bien sûr ! Mais il y a de bonnes et de mauvaises façons de le faire.

La mauvaise approche est de simplement mettre l'algorithme principal dans une classe. Ce qui donnerait :

```
class LièvreTortue
    constructor LièvreTortue
    end

    method jouer
        // Idem algorithme jeuLièvreTortue() ci-avant
    end
end
```

Ce qui réduirait l'algorithme principal à :

```
algorithm jeuLièvreTortue
    jeu: LièvreTortue
    jeu = new LièvreTortue()
    jeu.jouer()
end
```

ou même, en se passant de la variable locale :

```
algorithm jeuLièvreTortue
    (new LièvreTortue()).jouer()
end
```

Cette approche est correcte mais n'exploite en rien les avantages de l'OO. Une meilleure idée est de suivre l'approche MVC que nous allons vous expliquer.

Dans l'approche MVC, on découpe le code en différentes parties. La partie « modèle » regroupe les bouts de code qui font vraiment quelque chose (on parle de « métier ») tandis que la partie « vue » regroupe les bouts de code qui interagissent avec l'utilisateur (demandes et affichages). La partie « contrôleur », quant à elle, conserve le code qui fait le lien entre le modèle et la vue.

Si on respecte cette approche, le métier ne contient **aucune** interaction avec l'utilisateur et la vue ne s'occupe **que** de l'interaction avec l'utilisateur. Il y a là de nombreux avantages :

- ▷ Les compétences pour écrire le modèle (connaissance du métier, accès à des bases de données...) et le dialogue avec les utilisateurs (ergonomie, graphisme...) ne sont pas les mêmes. On peut donc confier ces parties à des équipes spécialisées.
- ▷ On pourra facilement changer le dialogue avec l'utilisateur. Ainsi, si on possède une version console du jeu, il suffira, pour en faire une autre version (console, graphique, web...) de recommencer la vue (et probablement d'adapter le contrôleur) sans toucher au modèle.

**Le modèle.** Dans notre exemple, le modèle contiendrait les algorithmes suivants :

- ▷ Initialiser le jeu : placer la tortue et le lièvre à leur position de départ.
- ▷ Jouer un coup : lancer le dé et déplacer le lièvre ou la tortue.
- ▷ Tester si le jeu est fini ou pas.
- ▷ Trouver le vainqueur.

Chacun de ces algorithmes est implémenté par une méthode (sauf l'initialisation qui est du ressort du constructeur). Au niveau des attributs, on retrouve les éléments du jeu : le lièvre, la tortue et le dé. [6]

Ce qui donne :

```
class LièvreTortue
    private :
        tortue: Tortue
        lièvre: Lièvre
        dé: Dé
    public :
        constructor LièvreTortue
        method estFini() → boolean
        method jouerCoup()
        method getVainqueur() → string
        //
        + les accesseurs (getLièvre(), getTortue() et getDé()) des attributs mais pas les mutateurs
end
```

6. Une autre façon de voir les choses est de dire qu'on trouve en attributs les variables locales de la version non OO qui sont partagées par les différentes méthodes. Ici, il s'agit de toutes les variables locales mais ce n'est pas toujours le cas. Le dé, par exemple, n'est un attribut que parce que la vue voudra le connaitre pour le montrer à l'utilisateur. Sans cela, il pourrait être une variable locale de la méthode jouerCoup().

```
constructor LièvreTortue
    tortue = new Tortue()
    lièvre = new Lièvre()
    dé = new Dé(6)
end

method estFini() → boolean
    return tortue.estArrivée() OU lièvre.estArrivé()
end
```

```
method jouerCoup
    dé.lancer()
    if dé.getValeur()=6
        lièvre.avancer()
    end
    tortue.avancer()
end

method getVainqueur() → string
    if lièvre.estArrivé()
        return "Lièvre"
    else if tortue.estArrivée()
        return "Tortue"
    else
        return "Aucun, partie en cours"
    end
end
```

Remarque : Dans un langage comme JAVA, la méthode getVainqueur() pourrait lancer une `IllegalStateException` si elle est appelée avant que la partie ne soit finie.

**La vue.** Avec notre exemple, plutôt simple, il n'y a pas de lecture mais il reste des affichages :

- ▷ Afficher l'état du jeu après un coup : valeur du dé et nouvelles positions du lièvre et de la tortue.
- ▷ Afficher le vainqueur.

Cet exemple est probablement trop simple pour nécessiter une classe [7]. On pourrait se contenter de deux algorithmes classiques.

```
algorithm afficherÉtat(jeu: LièvreTortue)
    print jeu.getDé().getValeur()
    print jeu.getTortue().getAvancée()
    print jeu.getLièvre().estArrivé()
end

algorithm afficherVainqueur(jeu: LièvreTortue)
    print "Le gagnant est : ", jeu.getVainqueur()
end
```

7. Une erreur classique est de placer ces algorithmes dans les classes associées du modèle car cela contrevient à la règle : aucune interaction utilisateur dans le modèle.

**Le contrôleur.** Le contrôleur est ce qui n'a pas été placé dans la vue ou le modèle, c'est-à-dire le code qui crée la dynamique entre tous ces éléments.

```
algorithm jeuLièvreTortue
    jeu: LièvreTortue
    jeu = new LièvreTortue()
    while NON jeu.estFini()
        jeu.jouerCoup()
        afficherÉtat(jeu)
    end
    afficherVainqueur(jeu)
end
```

Remarquez la **concision** et la **lisibilité** de ce qu'on vient d'écrire. C'est quasiment ce qu'on avait écrit comme premier jet lorsqu'on a abordé ce problème au tout début.

En Java :

```java
package dev2.algo.oo.lievretortue;

/**
 * Partie "modèle" de la version MVC du jeu du lièvre et de la tortue.
 */
public class LievreTortue {

    private final Tortue tortue;
    private final Lievre lièvre;
    private final De dé;

    public LievreTortue() {
        tortue = new Tortue();
        lièvre = new Lievre();
        dé = new De();
    }

    public Tortue getTortue() {
        return tortue;
    }

    public Lievre getLièvre() {
        return lièvre;
    }

    public De getDé() {
        return dé;
    }

    public void jouerCoup() {
        dé.lancer();
        if (dé.getValeur() == 6) {
            lièvre.avancer();
        } else {
            tortue.avancer();
        }
    }

    public boolean estFini() {
        return tortue.estArrivée() || lièvre.estArrivé();
    }

    public String getVainqueur() {
        if (lièvre.estArrivé()) {
            return "lièvre";
        } else {
            return "tortue";
        }
    }
}
```

```java
package dev2.algo.oo.lievretortue;

/**
 * Partie "vue" de la version MVC du jeu du lièvre et de la tortue.
 */
public class Vue {

   public static void afficherÉtat(LievreTortue jeu) {
      System.out.println("dé=" + jeu.getDé().getValeur()
            + ", tortue=" + jeu.getTortue().getAvancée()
            + ", lièvre=" + (jeu.getLièvre().estArrivé() ? 6 : 0));
   }

   public static void afficherVainqueur(LievreTortue jeu) {
      System.out.println("Le vainqueur est: " + jeu.getVainqueur());
   }

}
```

```java
package dev2.algo.oo.lievretortue;

/**
 * Partie "contrôleur" de la version MVC du jeu du lièvre et de la tortue.
 */
public class JeuLievreTortue {

   public static void main(String[] args) {
      LievreTortue jeu = new LievreTortue();
      while (!jeu.estFini()) {
         jeu.jouerCoup();
         Vue.afficherÉtat(jeu);
      }
      Vue.afficherVainqueur(jeu);
   }
}
```