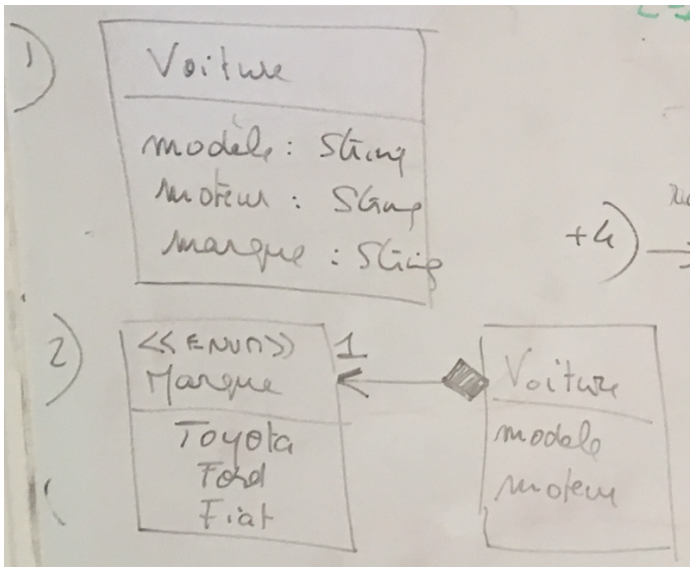
CORRIGES
TD
Diagramme de classes, d'objets et de séquence
Diagramme de classes métier
Design pattern

Sommaire

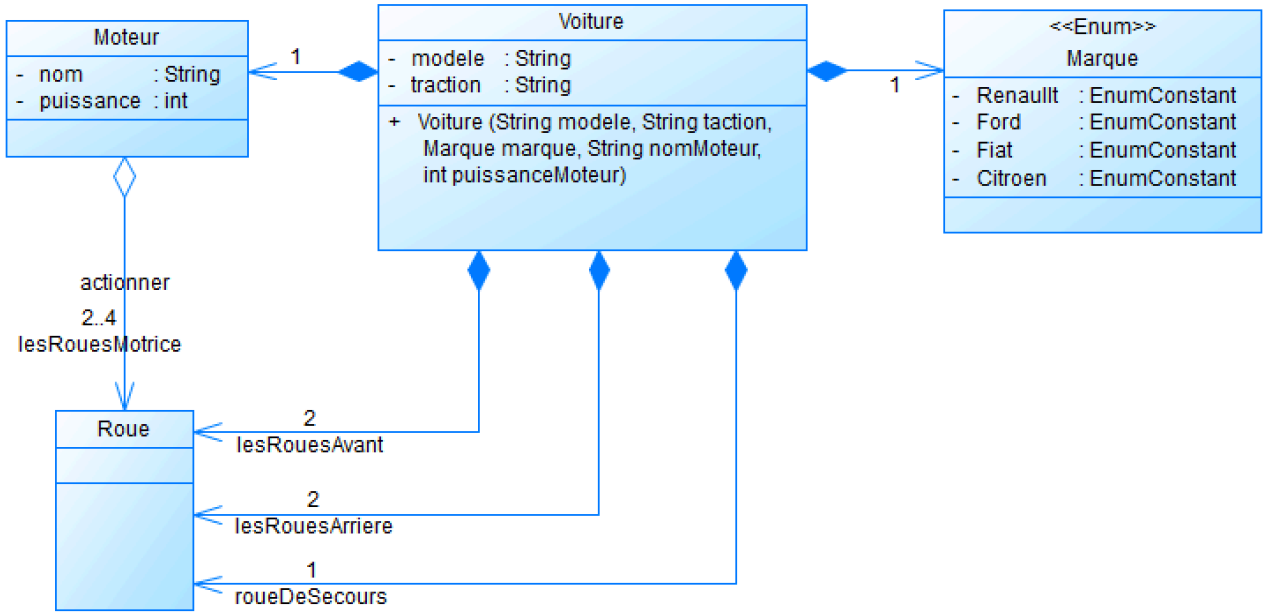
1 - Diagramme de classe

1 : La voiture (d'après L. Audibert – Ellipses)

1. Une voiture a une marque, un modèle, un moteur. Modéliser la situation.
2. La marque appartient à une liste que vous choisirez (énumération). Modéliser la situation.



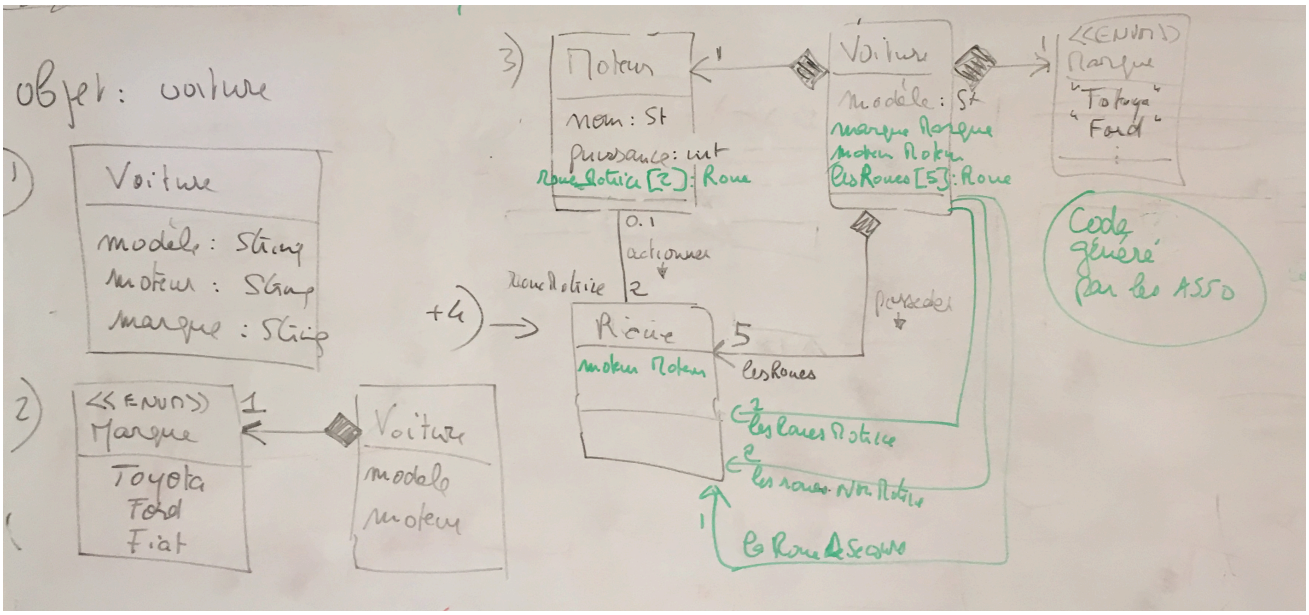
3. Le moteur a un nom et une puissance. Modéliser la situation.
4. La voiture a des roues (avant, arrière et de secours) et le moteur actionne 2 de ces roues. Modéliser la situation.
5. On souhaite préciser si le moteur actionne les roues avant ou les roues arrières. Modéliser la situation.



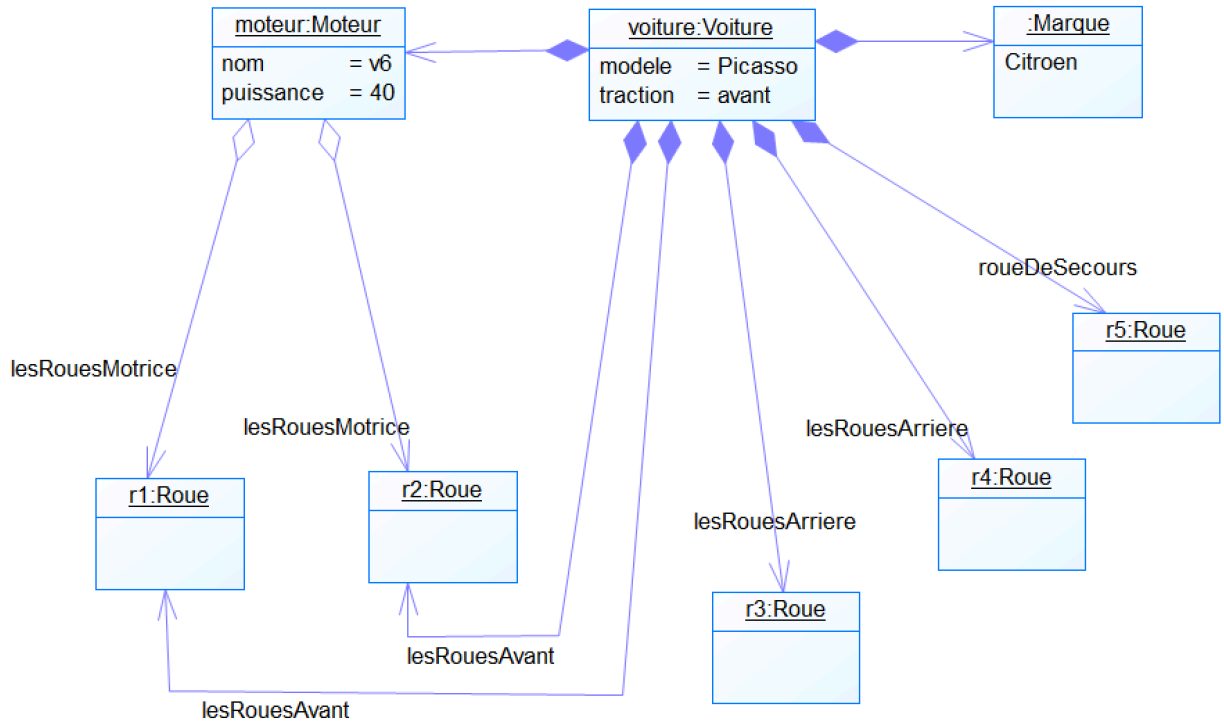
Ici, la voiture a 2 roues avant, 2 roues arrière et une roue de secours. Le moteur a des roues motrices (2 ou 4). Le constructeur se chargera d'associer les bonnes roues à la voiture en fonction du type de traction.

Sur ce diagramme, on voit en vert, dans les classes, le code généré par les association.

Les associations en verts (lesRouesMotrices, lesRouesNonMotrices et laRoueDeSecours) ne sont pas prises en comptes : on en est au stade où on a une seule association : lesRoues, avec 5 éléments.



6. Faire un diagramme d'objets correspondant diagramme de classes précédent pour un objet de la classe voiture à traction avant.



7. Ecrire le constructeur de la voiture sous la forme d'un diagramme de séquence. On peut aussi coder le constructeur (en plus du diagramme de séquence) si on préfère.

```
public class Voiture {
   private String modele;
   private String traction;
   private Marque marque;
   private Moteur moteur;
   private Roue roueDeSecours;
   private java.util.List<Roue> lesRouesAvant;
   private java.util.List<Roue> lesRouesArriere;

   public Voiture(String modele,    String taction,      Marque marque,
                  String nomMoteur, int puissanceMoteur) {
      this.modele=modele;
      this.traction=traction;
      this.marque=marque;
      this.moteur=new Moteur(nomMoteur, puissanceMoteur);
      this.lesRouesAvant=setLesRouesAvant(newRoue(), newRoue());
      this.lesRouesArriere=setLesRouesArriere(newRoue() newRoue());
      this.roueDeSecours=newRoue();
      if traction = "avant"
         this.moteur.setRouesMotrice(this.getRouesAvant());
      else if traction = "arriere"{
         this.moteur.setRouesMotrice(this.getRouesArriere());
      }

      else if traction = "4x4"{
         this.moteur.setRouesMotrice(this.getRouesAvant());
         this.moteur.setRouesMotrice(this.getRouesArriere());
      }
   }
}
```

En détaillant le constructeur, on constate la nécessité d'avoir des méthodes en plus :

Dans la Classe Voiture

setLesRouesAvant : pour mettre directement deux roues avant dans « lesRouesAvant »

setLesRouesArriere : pour mettre directement deux roues arrière dans « lesRouesArriere »

Dans la classe Moteur :

setRouesMotrices : pour mettre des roues avant ou arrière ou les deux comme roues motrices associées au moteur.

Du coup, il faut ajouter dans la classe Moteur :

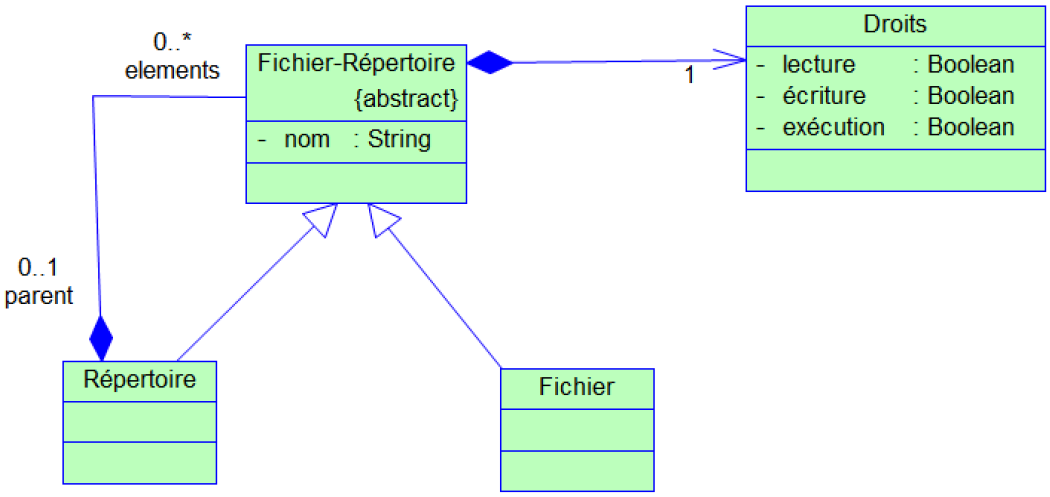
getRouesAvant()

getRouesArriere()

A noter qu'avec le test sur la traction (avant, arriere, 4x4), on voit qu'on ferait mieux de définir un énuméré Traction.

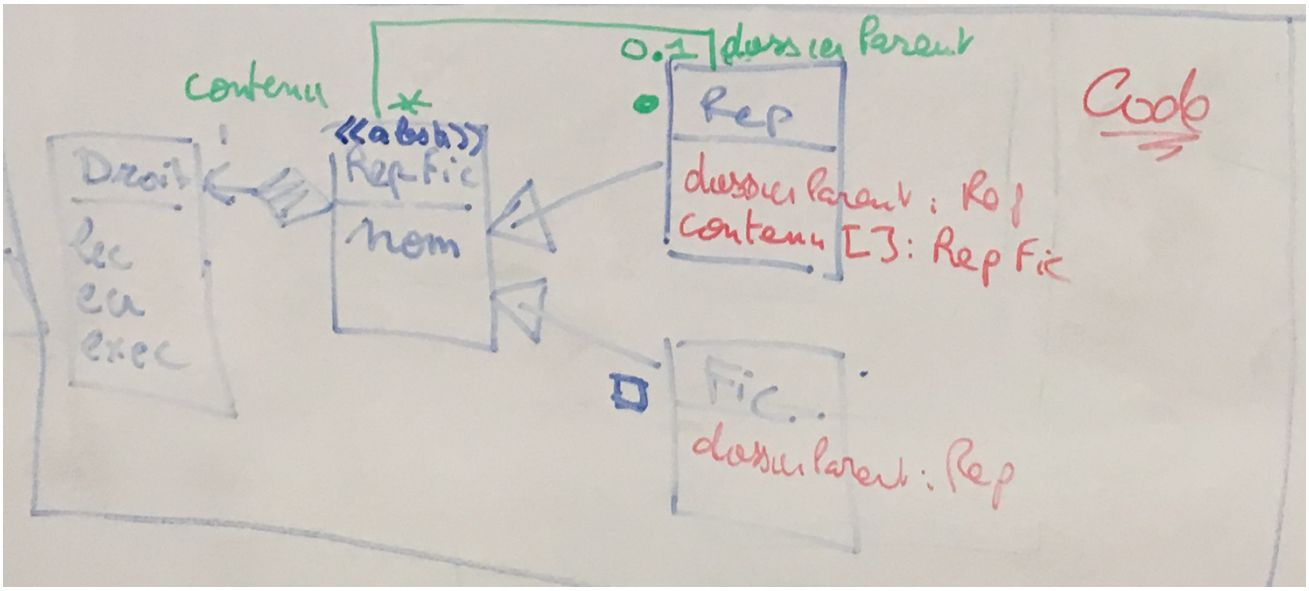
2 : Système de fichiers (d'après L. Audibert – Ellipses)

1. Un répertoire possède un nom et des droits en lecture, exécution et écriture. Modéliser la situation.
2. Un fichier a les mêmes caractéristiques. Mettre à jour le modèle précédent.
3. Un répertoire peut contenir des fichiers et des répertoires. Tous les fichiers sont dans un répertoire et un seul. Tous les répertoires sont dans un répertoire et un seul sauf le répertoire racine qui n'est dans aucun répertoire. Modélisez la situation.



Dans le schéma ci-dessous, en rouge : le code généré par l'association en vert.

Déta



Détail du code :

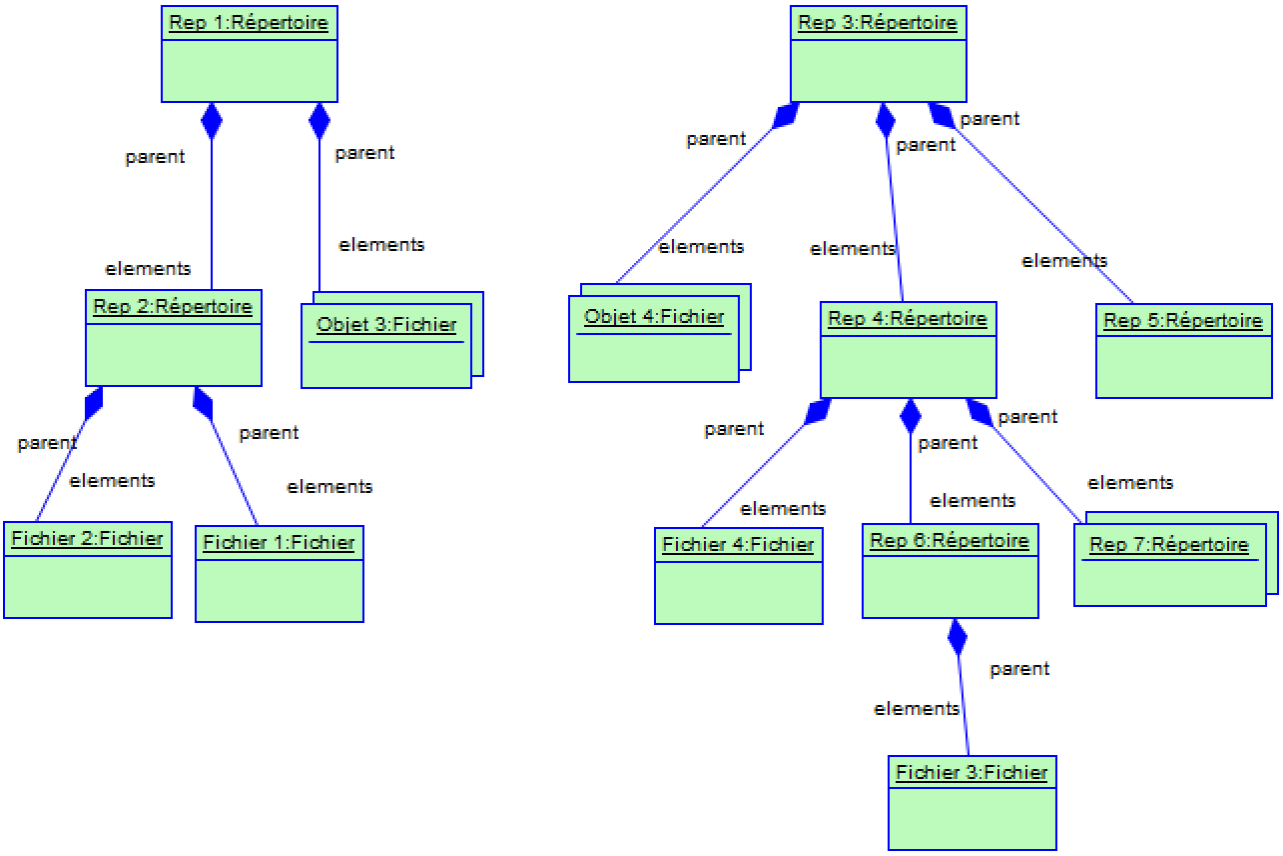
```
public abstract class FichierRepertoire {
  protected String nom;
  protected Repertoire parent;
  protected Droits droits;
}

public class Repertoire extends FichierRepertoire {
  private FichierRepertoire[] elements;
}

public class Fichier extends FichierRepertoire {
}

public class Droits {
  private Boolean lecture;
  private Boolean ecriture;
  private Boolean execution;
}
```

4. Proposer un diagramme d'objets avec deux arborescences distinctes de hauteur 2 et 3, et avec des fichiers.

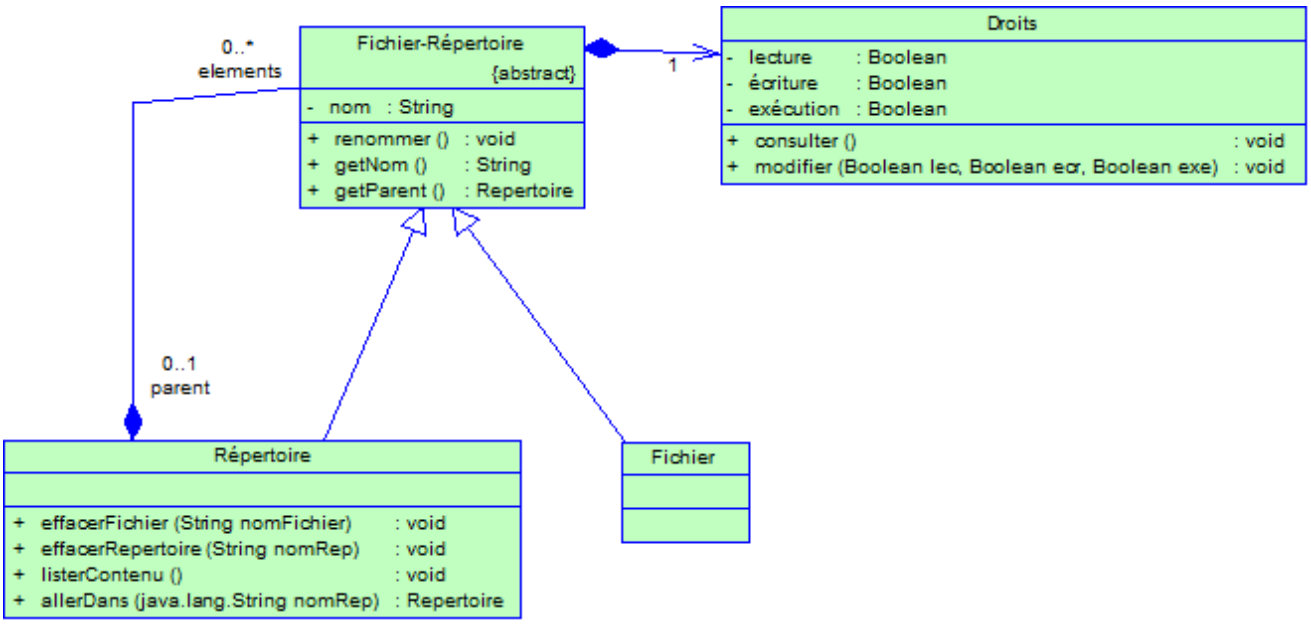


5. On peut effacer et renommer les fichiers et les répertoires. A partir d'un répertoire, on peut remonter à son parent, on peut lister son contenu, on peut se rendre dans un sous répertoire. Modélisez la situation.

« **renommer** » est une méthode qu'on définit au niveau du « fichier-répertoire ». On y placera le code.
« effacer » (c'est-à-dire « supprimer ») doit être séparée en 2 cas : effacer un ficher et effacer un répertoire. Dans tous les cas, l'effacement se fait à partir du d'un répertoire : ce sont donc des méthodes qu'on trouve dans la classe Répertoire.
« **listerContenu** » et « **allerDans** » sont des méthodes du Répertoire.
« **getParent** » est une méthode commune, définit au niveau du « fichier-répertoire ». On y placera le code.

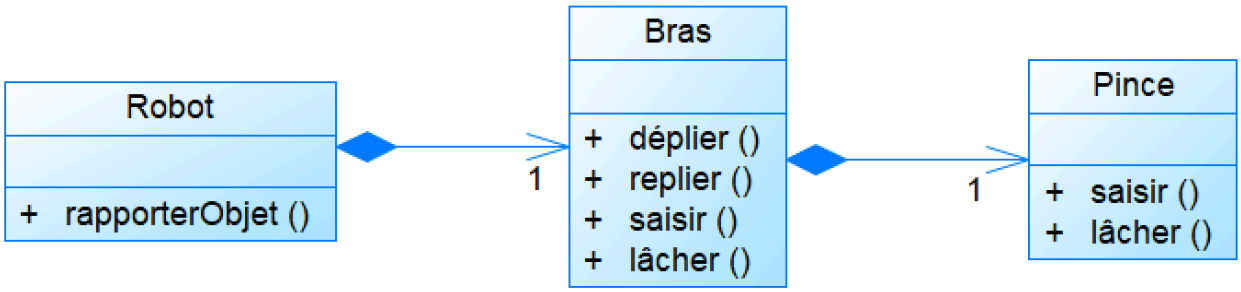
6. On peut consulter ou modifier les droits d'un fichier ou d'un répertoire. Le principe est qu'on modifie les trois droits en même temps selon une syntaxe particulière. On veut aussi pouvoir accéder au nom d'un fichier ou d'un répertoire. Modélisez la situation.

« **consulter**» et « **modifier**» sont des méthodes qu'on définit au niveau des Droits
« **getNom** » est une méthode commune, définit au niveau du « fichier-répertoire ». On y placera le code.



3 : Le robot

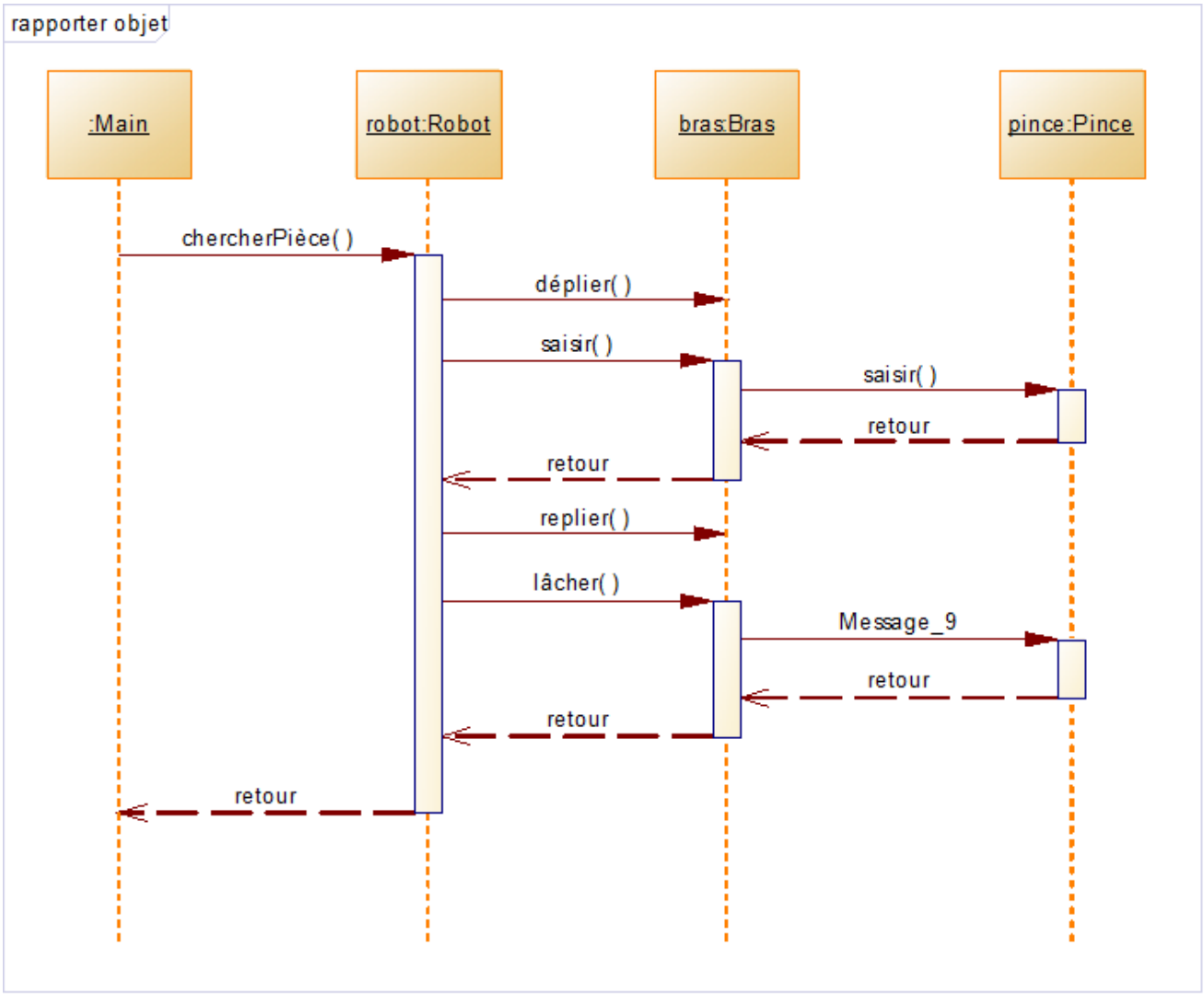
Soit un robot modélisé comme suit :



Le robot qui dispose d'un bras articulé se terminant par une pince. Le robot peut rapporter des objets.

1. Faire le diagramme de séquence permettant de décrier cette méthode. On considère que pour rapporter un objet le robot doit déplier son bras, puis saisir l'objet avec sa pince, puis replier son bras et enfin lâcher l'objet.

   Sur un logiciel, on commencera par reproduire le diagramme de classes pour pouvoir l'utiliser pour le diagramme de classe.



4 : Jeu de rôles et tournevis !

1)

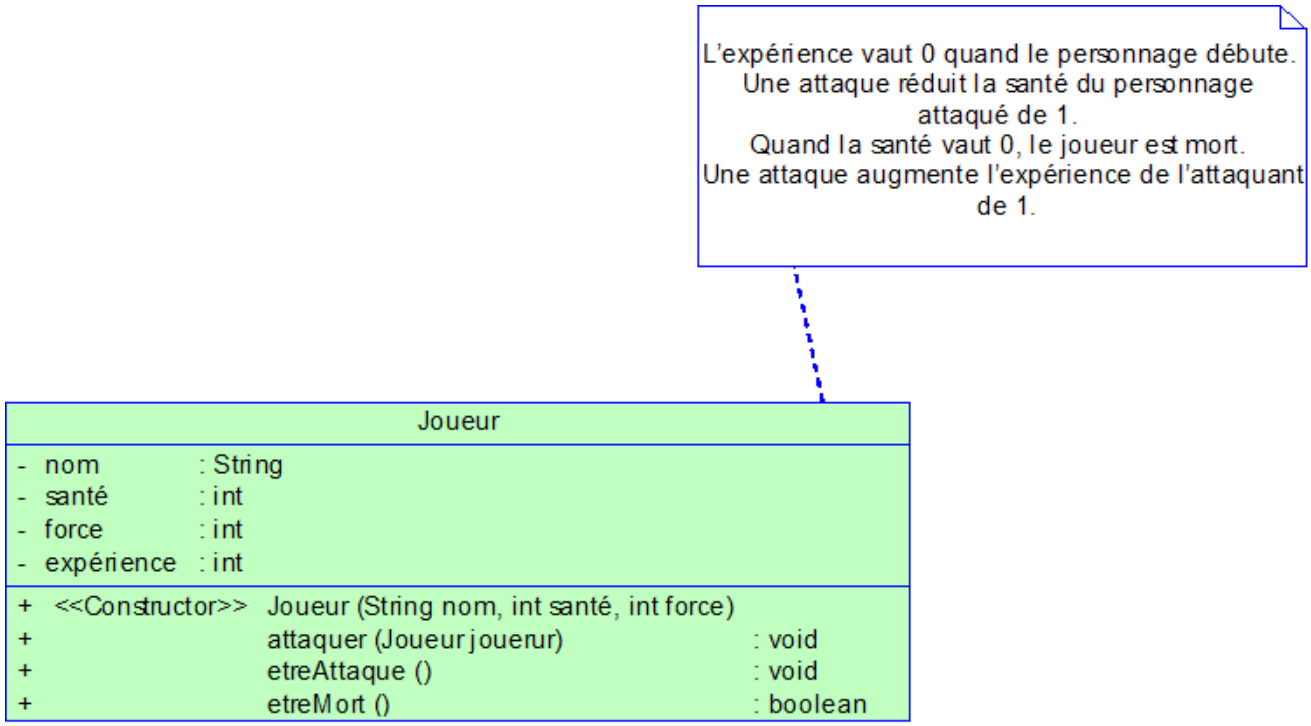
Soit un jeu vidéos avec des joueurs. Les joueurs ont un nom, une santé, une force et une expérience. Ces trois dernières caractéristiques sont des entiers positifs ou nuls. L'expérience vaut 0 quand le personnage débute.

On se dote d'un constructeur qui permet de donner un nom, une santé et une force au joueur.

On peut afficher les caractéristiques d'un joueur (son nom et la valeur de tous ses attributs).

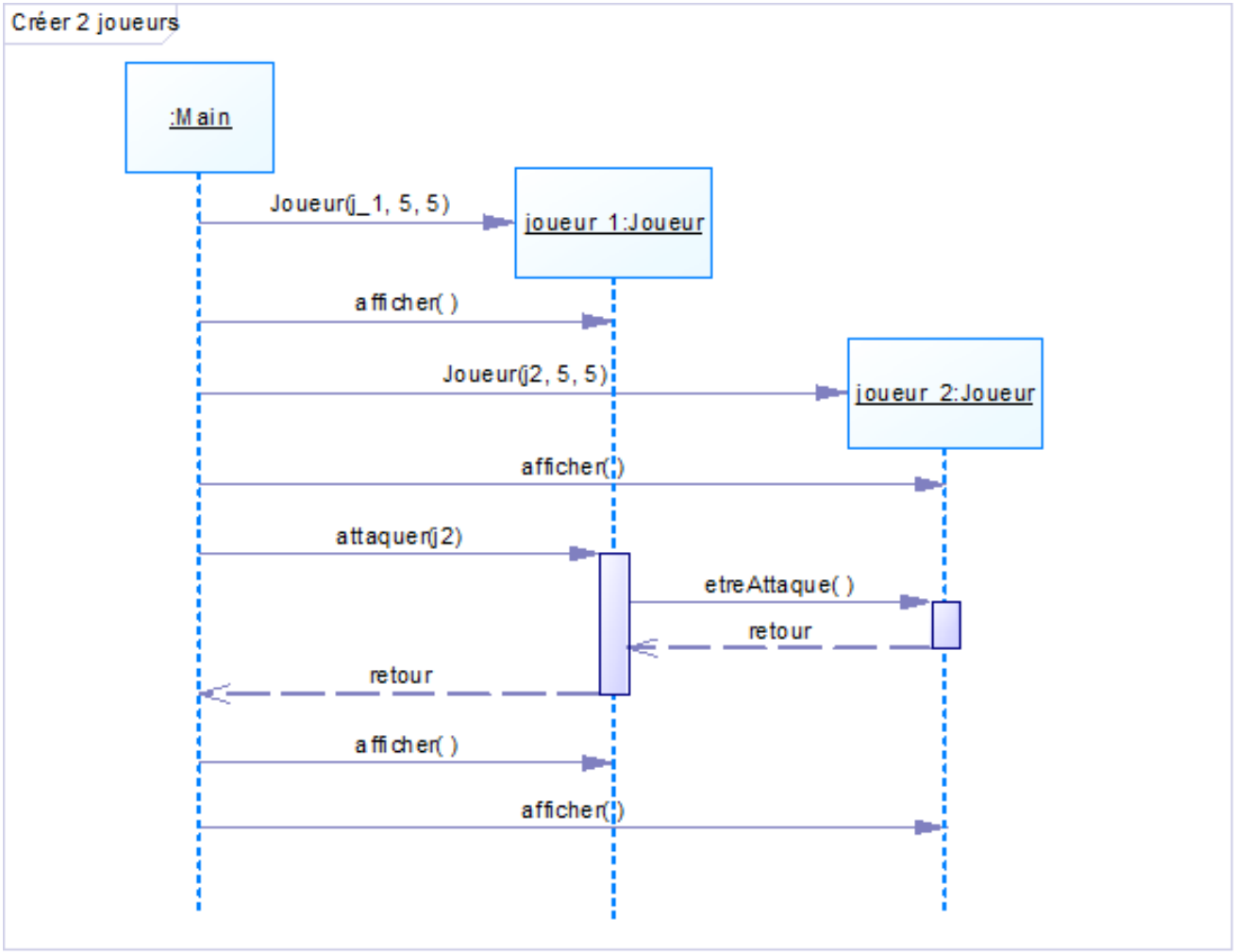
Un joueur peut attaquer un autre joueur. Une attaque réduit la santé du personnage attaqué de 1. Quand la santé vaut 0, le joueur est mort. Une attaque augmente l'expérience de l'attaquant de 1.

1. Définir la classe, les attributs et les méthodes qui permettent de gérer cette situation.



Le commentaire est attaché à la classe. Il permet de préciser des éléments qu'on retrouvera dans le code.

2. Ecrire un *main* sous la forme d'un diagramme de séquence qui déclare deux joueurs et affiche leur caractéristiques. Ensuite le main fait attaquer le deuxième joueur par le premier. Afficher les nouvelles caractéristiques des joueurs. On peut aussi coder le main (en plus du diagramme de séquence) si on préfère.



2)

On crée maintenant des ennemis. Les ennemis ont les mêmes caractéristiques que les joueurs mais ils n'ont pas d'expérience et ils ont une race (troll, elfe, sorcier, etc.), et une valeur qui est un entier positif. Quand un joueur attaque un ennemi, il gagne en expérience la valeur de l'ennemi. Quand un ennemi attaque un personnage, il lui fait perdre 1 point de santé.

3. Définir les classes, les attributs et les méthodes qui permettent de gérer cette situation.

Diagramme de classe détaillé :

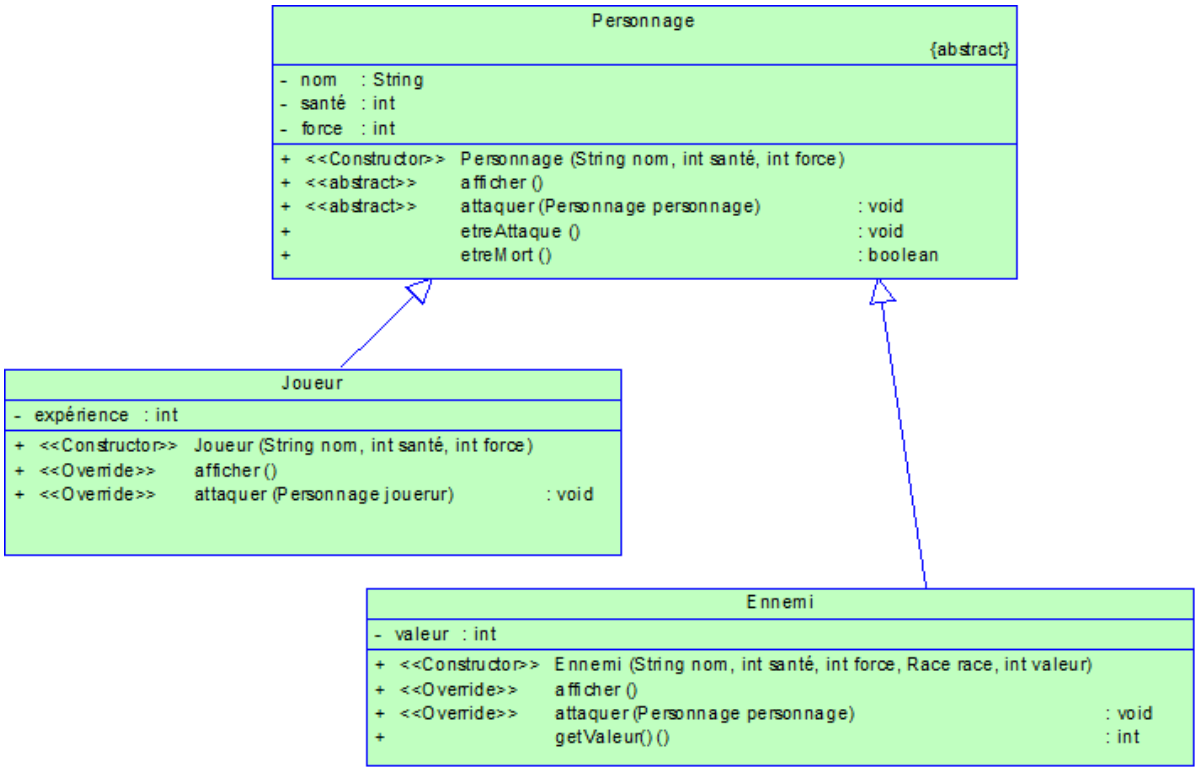
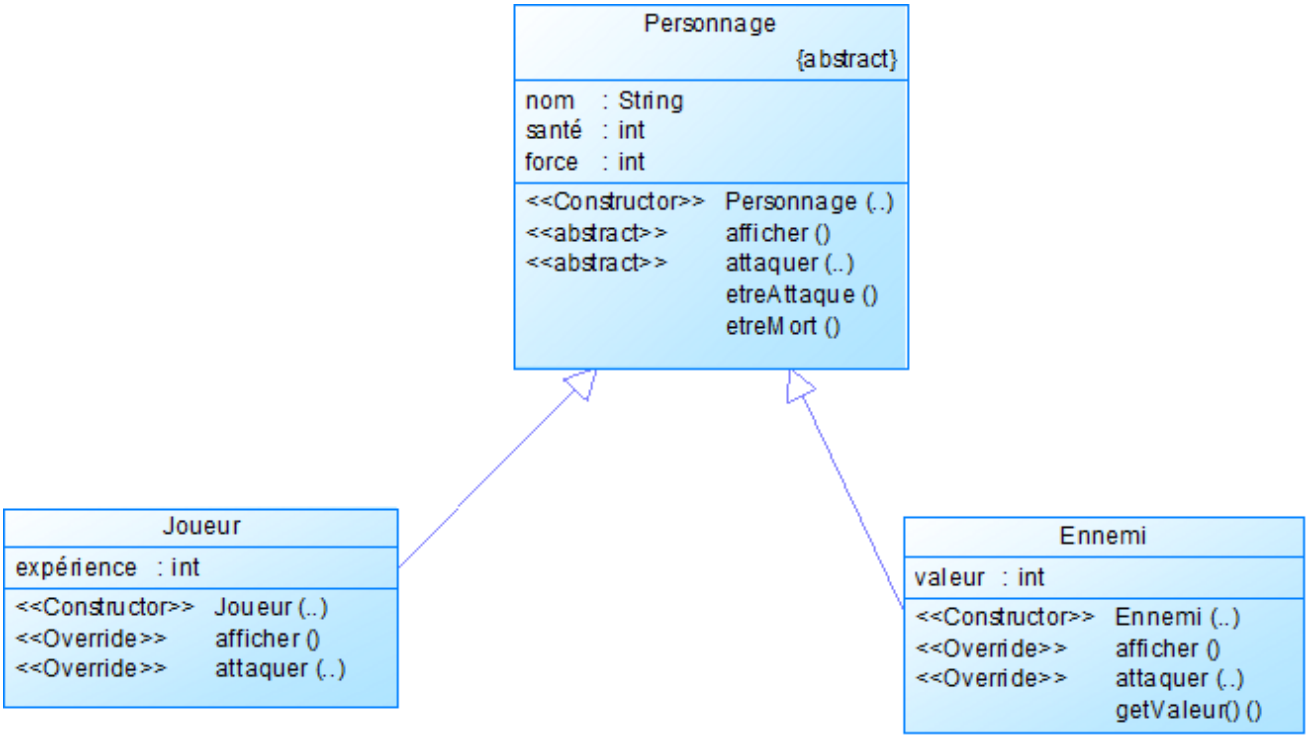
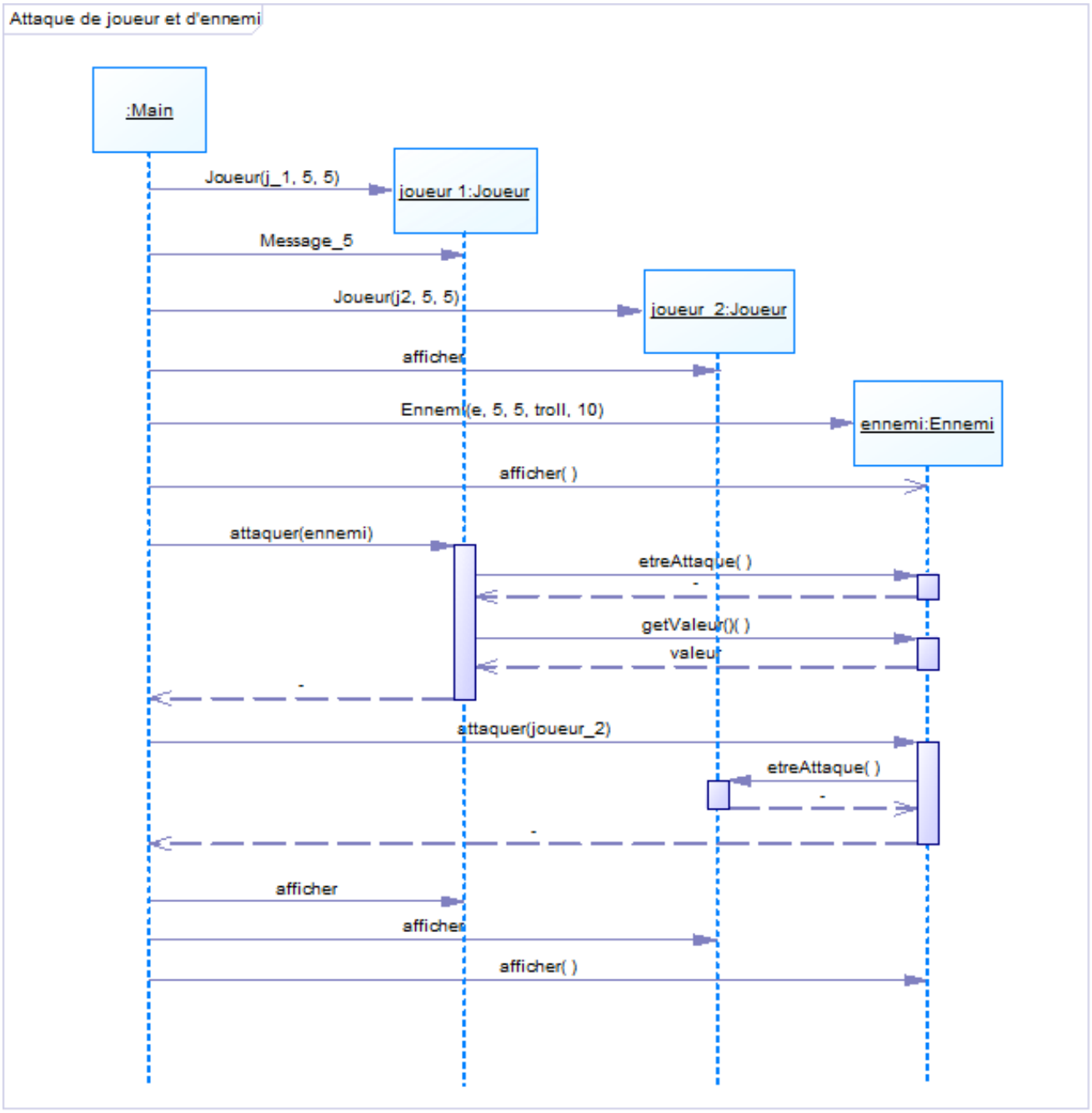


Diagramme de classe moins détaillé :



4. Ecrire un *main* sous la forme d'un diagramme de séquence qui déclare deux joueurs et un ennemi, affiche leur caractéristiques. Ensuite le main fait attaquer l'ennemi par le premier joueur. Puis le main fait attaquer le deuxième joueur par l'ennemi. Afficher les nouvelles caractéristiques de l'ennemi et des joueurs. On peut aussi coder le main (en plus du diagramme de séquence) si on préfère.



3)

Dans le jeu, les joueurs peuvent utiliser des outils. Ils peuvent trouver ou gagner des outils qui sont alors à leur disposition.

Un outil a un poids et un nom. On peut afficher les caractéristiques de l'outil.

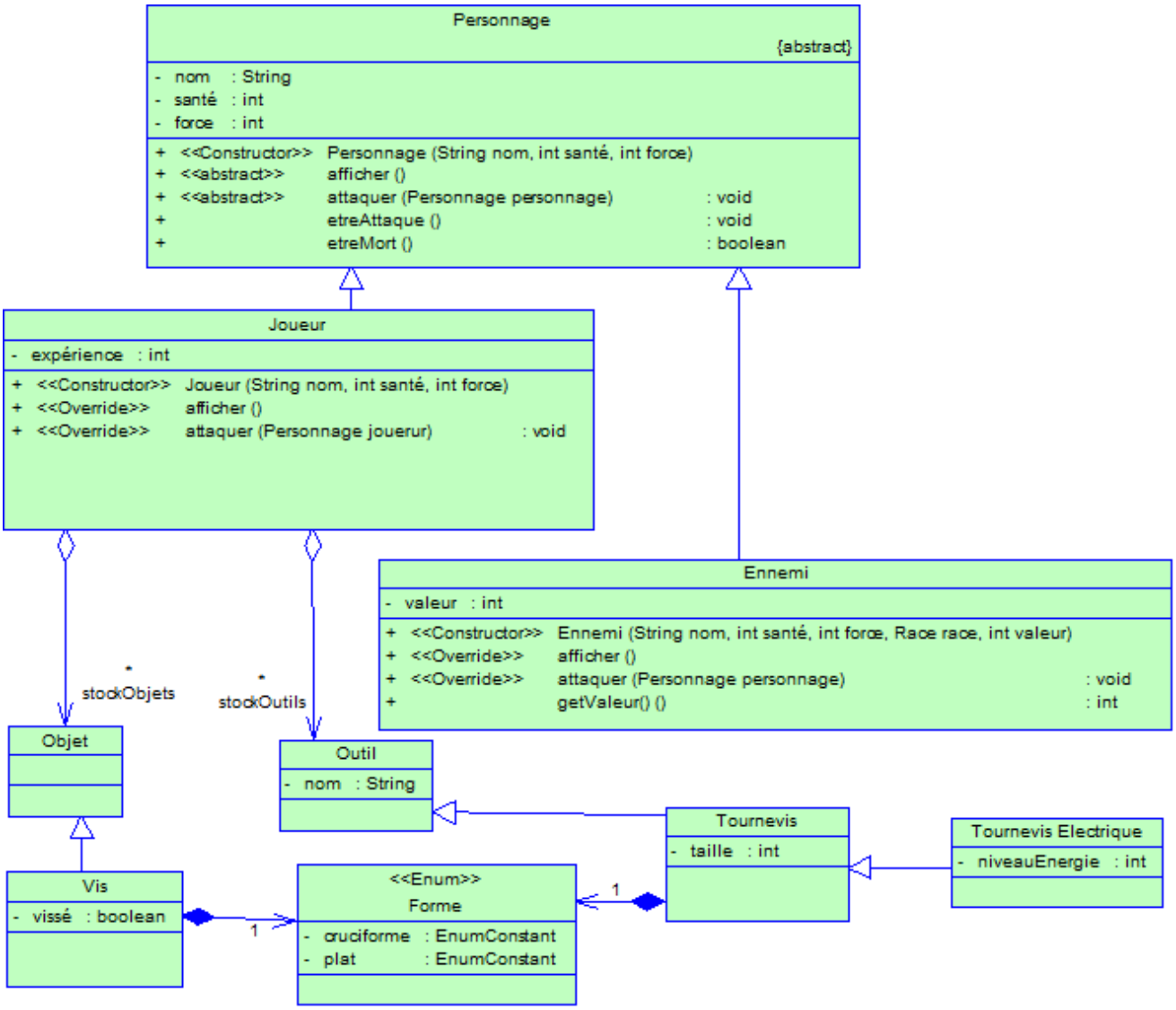
Le tournevis est un outil qui a une forme particulière (cruciforme, plat, etc.) avec une taille donnée en entier.

Les tournevis permettent de visser et de dévisser des vis. Une vis est un objet qui à une forme et une taille comme le tournevis. Ainsi, un tournevis peut être adapté ou pas à une vis. Une vis peut être vissée ou dévissée (dans ce cas elle est disponible pour être vissée). Dans le jeu, les joueurs peuvent trouver ou gagner des vis qui sont alors à leur disposition.

Le joueur peut chercher une vis dans son stock qui correspond à un de ses tournevis et inversement chercher un tournevis qui correspond à une vis.

Il existe des tournevis électriques qui ont une batterie avec un niveau d'énergie. Pour visser ou dévisser, ces tournevis doivent avoir de la batterie.

5. Définir toutes les classes nécessaires pour gérer la situation.



6. Ecrire un *main* sous la forme d'un diagramme de séquence qui qui permet à un joueur de stocker un tournevis électrique et une vis qu'il a trouvé. Ensuite le joueur choisit une vis puis un tournevis électrique adapté pour cette vis et visse cette vis (on ne s'intéresse pas à savoir sur quoi il la visse !). On peut aussi coder le main (en plus du diagramme de séquence) si on préfère.