Implémentation OO

Java

1 But

Le but de ces exercices est de mettre en pratique l'implémentation des concepts de base de l'orienté-objet en Java.

Parfois, il s'agira également de réaliser l'analyse et la conception avant d'en faire l'implémentation.

2 Prérequis

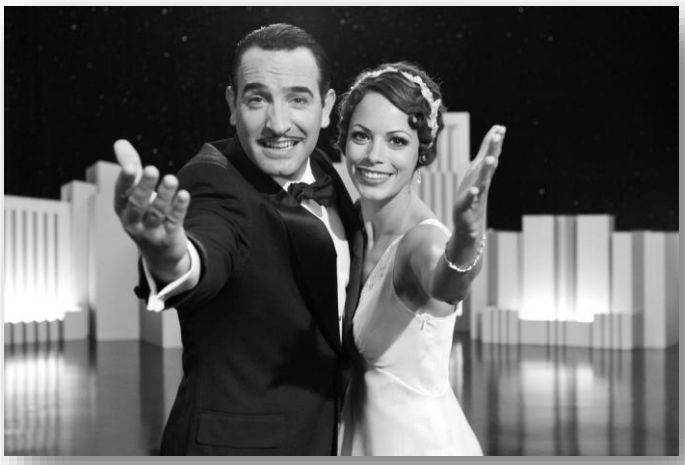
Connaître le diagramme de classe en UML.

3 Travail à réaliser

3.1 L'individu

Soit le diagramme de classes suivant :

Individu
+nom : String +prénom : String +annéeNaissance : int
+présenter() : String



Implémentez cette classe en Java.

La méthode `présenter()` doit retourner une chaîne de caractères de ce format :

```
Je m'appelle prénom nom et je suis né(e) en annéeNaissance.
```

Ensuite, créez une classe `Application` contenant la méthode statique `main()` et instanciez les objets selon ce diagramme d'objets :

actrice:Individu
nom : Bejo prénom : Bérénice annéeNaissance : 1976

acteur:Individu
nom : Dujardin prénom : Jean annéeNaissance : 1972

Les faire se présenter.

Affichez ensuite celui qui est le plus jeune.

Voici un exemple d'affichage :

```
Je m'appelle Jean Dujardin. Je suis né(e) en 1972.
Je m'appelle Bérénice Bejo. Je suis né(e) en 1976.
L'individu le plus jeune : Bejo
```

Individu.java

```
/**
 * Classe représentant un individu.
 * @author conujer
 */
package ex01;

public class Individu {
	public String nom;
	public String prénom;
	public int annéeNaissance;

	/**
	 * @return une chaîne de caractères contenant la présentation de l'individu.
	 */
	public String présenter() {
		return "Je m'appelle " + prénom + " " + nom + ". Je suis né(e) en " +
annéeNaissance + ".";
	}
}
```

Application.java

```
/**
 * Exercices OO : exercice 1.
 * @author conujer
 */
package ex01;

public class Application {

	/**
	 * Point d'entrée du programme.
	 */
	public static void main(String[] args) {
		// Création des individus
		Individu acteur = new Individu();
		acteur.nom = "Dujardin";
		acteur.prénom = "Jean";
		acteur.annéeNaissance = 1972;

		Individu actrice = new Individu();
		actrice.nom = "Bejo";
		actrice.prénom = "Bérénice";
		actrice.annéeNaissance = 1976;

		// Présentation
		System.out.println(acteur.présenter());
		System.out.println(actrice.présenter());
```

```
            // Détermination du plus jeune
            Individu lePlusJeune = acteur;
            if (actrice.annéeNaissance > acteur.annéeNaissance)
                  lePlusJeune = actrice;

            System.out.println("L'individu le plus jeune : " + lePlusJeune.nom);
      }
}
```

3.2 L'individu, bis

À partir des classes créées à l'exercice 3.1, écrivez un programme qui réalise séquentiellement les opérations suivantes :

- Créer les acteurs *Jean Dujardin* et *Bérénice Bejo* ;
- Les faire se présenter ;
- Lassé de sa notoriété, *Dujardin* change de nom en *Dupotager*.
- *Bejo* choisi de prendre le nom de son mari. Elle s'appelle désormais *Hazanavicius*.
- Suite à ces changements, ces acteurs se présentent.

Voici un exemple d'affichage :

```
Je m'appelle Jean Dujardin. Je suis né(e) en 1972.
Je m'appelle Bérénice Bejo. Je suis né(e) en 1976.
Nous changeons de noms.
Je m'appelle Jean Dupotager. Je suis né(e) en 1972.
Je m'appelle Bérénice Hazanavicius. Je suis né(e) en 1976.
```

Individu.java

```
/**
 * Classe représentant un individu.
 * @author conujer
 */
package ex02;

public class Individu {
      public String nom;
      public String prénom;
      public int annéeNaissance;

      /**
       * @return une chaîne de caractères contenant la présentation de l'individu.
       */
      public String présenter() {
            return "Je m'appelle " + prénom + " " + nom + ". Je suis né(e) en " +
annéeNaissance + ".";
      }
}
```

Application.java

```
package ex02;

public class Application {

      /**
```

```
        * Point d'entrée du programme.
        */
       public static void main(String[] args) {
              Individu acteur = new Individu();
              acteur.nom = "Dujardin";
              acteur.prénom = "Jean";
              acteur.annéeNaissance = 1972;

              Individu actrice = new Individu();
              actrice.nom = "Bejo";
              actrice.prénom = "Bérénice";
              actrice.annéeNaissance = 1976;

              System.out.println(acteur.présenter());
              System.out.println(actrice.présenter());

              System.out.println("Nous changeons de noms.");

              acteur.nom = "Dupotager";
              actrice.nom = "Hazanavicius";

              System.out.println(acteur.présenter());
              System.out.println(actrice.présenter());
       }
}
```

3.3 L'individu, bis repetita

En reprenant les classes de l'exercice 3.1, vous décidez d'améliorer leur robustesse en encapsulant les attributs privés.

Modifiez la classe `Individu` de telle sorte que les attributs `nom`, `prénom` et `annéeNaissance` ne puissent plus être directement modifiés, mais uniquement au travers de mutateurs (méthodes `set`) et accesseurs (méthodes `get`).

Les mutateurs devront valider les données fournies avant de modifier les attributs de la classe :

- Les noms et prénoms doivent contenir au moins un caractère.
- L'année de naissance ne doit pas être inférieure à 1900 ni supérieure à 2200.

En cas de donnée invalide, aucun changement n'est effectué et un message d'erreur est inscrit sur la console de sortie.

Voici un exemple d'affichage :

```
Je m'appelle Jean Dujardin. Je suis né(e) en 1972.
Je m'appelle Bérénice Bejo. Je suis né(e) en 1976.
Le prénom doit contenir au moins un caractère.
Le nom doit contenir au moins un caractère.
2300 n'est pas une année valide.
Je m'appelle Jean Dujardin. Je suis né(e) en 1972.
Je m'appelle Bérénice Bejo. Je suis né(e) en 1976.
L'individu le plus jeune : Bejo
```

Individu.java

```
/**
 * Classe représentant un individu.
 * @author conujer
```

```
 */
package ex03;

public class Individu {
	private String nom;
	private String prénom;
	private int annéeNaissance;

	/**
	 * Change le nom de cet individu.
	 * Aucun changement n'a lieu si le nom n'est pas valide.
	 * @param nom Nouveau nom
	 */
	public void setNom(String nom) {
		if (nom.length() >= 1)
			this.nom = nom;
		else
			System.out.println("Le nom doit contenir au moins un
caractère.");
	}

	/**
	 * @return le nom de cet individu.
	 */
	public String getNom() {
		return nom;
	}

	/**
	 * Change le prénom de cet individu.
	 * Aucun changement n'a lieu si le prénom n'est pas valide.
	 * @param prénom Nouveau prénom
	 */
	public void setPrénom(String prénom) {
		if (prénom.length() >= 1)
			this.prénom = prénom;
		else
			System.out.println("Le prénom doit contenir au moins un
caractère.");
	}

	/**
	 * @return le prénom de cet individu.
	 */
	public String getPrénom() {
		return prénom;
	}

	/**
	 * Change l'année de naissance de cet individu.
	 * Aucun changement n'a lieu si l'année n'est pas valide.
	 * @param annéeNaissance Nouvelle année de naissance
	 */
	public void setAnnéeNaissance(int annéeNaissance) {
		if (annéeNaissance >= 1900 && annéeNaissance <= 2200)
			this.annéeNaissance = annéeNaissance;
		else
```

```
                System.out.println(annéeNaissance + " n'est pas une année
valide.");
        }

        /**
         * @return l'année de naissance de cet individu.
         */
        public int getAnnéeNaissance() {
                return annéeNaissance;
        }

        /**
         * @return une chaîne de caractères contenant la présentation de l'individu.
         */
        public String présenter() {
                return "Je m'appelle " + prénom + " " + nom + ". Je suis né(e) en " +
annéeNaissance + ".";
        }
}
```

Application.java

```
/**
 * Exercices OO : exercice 3.
 * @author conujer
 */
package ex03;

public class Application {

        /**
         * Point d'entrée du programme.
         */
        public static void main(String[] args) {
                // Création des individus
                Individu acteur = new Individu();
                acteur.setNom("Dujardin");
                acteur.setPrénom("Jean");
                acteur.setAnnéeNaissance(1972);

                Individu actrice = new Individu();
                actrice.setNom("Bejo");
                actrice.setPrénom("Bérénice");
                actrice.setAnnéeNaissance(1976);

                // Présentation
                System.out.println(acteur.présenter());
                System.out.println(actrice.présenter());

                // Modification de leurs caractéristiques
                acteur.setPrénom("");
                acteur.setAnnéeNaissance(1800);
                actrice.setNom("");
                actrice.setAnnéeNaissance(2200);

                System.out.println(acteur.présenter());
                System.out.println(actrice.présenter());
```

```
            // Détermination du plus jeune
            Individu lePlusJeune = acteur;
            if (actrice.getAnnéeNaissance() > acteur.getAnnéeNaissance())
                  lePlusJeune = actrice;

            System.out.println("L'individu    le    plus    jeune    :    "    +
lePlusJeune.getNom());
      }
}
```

3.4 L'individu, encore et toujours

Complétez la classe `Individu` de l'exercice précédent afin d'y ajouter différents constructeurs :

un constructeur prenant un nom, un prénom et une année de naissance ;
un constructeur prenant un nom et un prénom, l'année de naissance étant fixée à 2000.

Attention à ne pas dupliquer inutilement le code.

Modifier la classe `Individu` de sorte que l'individu se présente en écrivant le code suivant :

```
Individu monIndividu = new Individu(…);
System.out.println(monIndividu);
```

Modifier la phrase de présentation pour qu'elle affiche ceci :

```
Je m'appelle prénom nom. Cette année, je fête mes x ans.
```

Toutefois, si l'individu n'est pas encore né, afficher :

```
Je m'appelle prénom nom. Je ne suis pas encore né(e).
```

Si l'individu est né cette même année, afficher :

```
Je m'appelle prénom nom. Je suis né(e) cette année !
```

Pour connaître l'année en cours, utiliser le code suivant (uniquement depuis *Java 8*) :

```
LocalDate maintenant = LocalDate.now();
int cetteAnnee = maintenant.getYear();
```

Pour tester tous les cas de figure possibles, créez les individus suivants :

L'acteur *Jean Dujardin*, né en 1972.
L'actrice *Bérénice Bejo*, née en 1976.
Aidan Clinton, né en 2016[1]..
La biologiste de fiction, *Alexa Komarova* née en 2051 (ou 2052 mais peu importe).

Voici un exemple d'affichage :

[1] Si cet exercice est fait à un autre moment que l'année 2016, utiliser un autre personnage, dont l'année de naissance correspond à l'année scolaire en cours.

```
Je m'appelle Jean Dujardin. Cette année, je fête mes 44 ans.
Je m'appelle Bérénice Bejo. Cette année, je fête mes 40 ans.
Je m'appelle Aidan Clinton. Je suis né(e) cette année !
Je m'appelle Alexa Komarova. Je ne suis pas encore né(e).
```

Individu.java

```
/**
 * Classe représentant un individu.
 * @author conujer
 */
package ex04;

import java.time.LocalDate;

public class Individu {
	private final static int ANNEE_PAR_DEFAUT = 2000;

	private String nom;
	private String prénom;
	private int annéeNaissance;

	/**
	 * Construit un individu.
	 * Par défaut, son année de naissance est fixée à 2000.
	 * @param nom 			Nom de l'individu.
	 * @param prénom 	Prénom de l'individu.
	 */
	public Individu(String nom, String prénom) {
		this(nom, prénom, ANNEE_PAR_DEFAUT);
	}

	/**
	 * Construit un individu.
	 * @param nom 				Nom de l'individu.
	 * @param prénom 			Prénom de l'individu.
	 * @param annéeNaissance 	Année de naissance
	 */
	public Individu(String nom, String prénom, int annéeNaissance) {
		setNom(nom);
		setPrénom(prénom);
		setAnnéeNaissance(annéeNaissance);
	}

	/**
	 * Change le nom de cet individu.
	 * Aucun changement n'a lieu si le nom n'est pas valide.
	 * @param nom Nouveau nom
	 */
	public void setNom(String nom) {
		if (nom.length() >= 1)
		 this.nom = nom;
		else
		 System.out.println("Le nom doit contenir au moins un caractère.");
	}

	/**
	 * @return le nom de cet individu.
	 */
```

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

	/**
	 * Change le prénom de cet individu.
	 * Aucun changement n'a lieu si le prénom n'est pas valide.
	 * @param prénom Nouveau prénom
	 */
	public void setPrénom(String prénom) {
		if (prénom.length() >= 1)
		 this.prénom = prénom;
		else
		 System.out.println("Le prénom doit contenir au moins un caractère.");
	}

	/**
	 * @return le prénom de cet individu.
	 */
	public String getPrénom() {
		return prénom;
	}

	/**
	 * Change l'année de naissance de cet individu.
	 * Aucun changement n'a lieu si l'année n'est pas valide.
	 * @param annéeNaissance Nouvelle année de naissance
	 */
	public void setAnnéeNaissance(int annéeNaissance) {
		if (annéeNaissance >= -1900 && annéeNaissance <= 2200)
		 this.annéeNaissance = annéeNaissance;
		else
		 System.out.println(annéeNaissance + " n'est pas une année valide.");
	}

	/**
	 * @return l'année de naissance de cet individu.
	 */
	public int getAnnéeNaissance() {
		return annéeNaissance;
	}

	/**
	 * @return une chaîne de caractères contenant la présentation de l'individu.
	 */
	 @Override
	public String toString() {
		String présentation = "Je m'appelle " + prénom + " " + nom + ".";
		if (getAge() > 0)
		 présentation += " Cette année, je fête mes " + getAge() + " ans.";
		else if (getAge() == 0)
		 présentation += " Je suis né(e) cette année !";
		else
		 présentation += " Je ne suis pas encore né(e).";
		return présentation;
	}

	/**
```

```
         * @return l'âge que cet individu aura cette année.
         */
        private int getAge() {
              LocalDate maintenant = LocalDate.now();
              return maintenant.getYear() - annéeNaissance;
        }
}
```

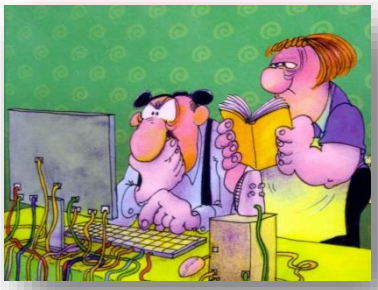
Application.java

```
/**
 * Exercices OO : exercice 4.
 * @author conujer
 */
package ex04;

public class Application {

    /**
     * Point d'entrée du programme.
     */
    public static void main(String[] args) {
      Individu acteur     = new Individu("Dujardin", "Jean", 1972);
      Individu actrice    = new Individu("Bejo", "Bérénice", 1976);
      Individu princesse  = new Individu("Clinton", "Aidan", 2016);
      Individu biologiste = new Individu("Komarova", "Alexa", 2051);

      System.out.println(acteur);
      System.out.println(actrice);
      System.out.println(princesse);
      System.out.println(biologiste);

    }
}
```

3.5 Le couple

Créez une classe Couple implémentant le diagramme de classes suivant :

Couple
-mari : String -épouse : String -annéeMariage : int
+Couple(mari : String, épouse : String, annéeMariage : int) +getMari() : String +getEpouse() : String +getAnnéeMariage() : int +toString() : String



La fonction `toString()` doit retourner une chaîne de caractères sous la forme suivante :

épouse et *mari* se sont mariés en *annéeMariage*.

Testez votre classe avec, par exemple, les couples suivants :
- *Pierre* et *Marie Curie*, mariés en 1895.
- *Michel Berger* et *France Gall*, mariés en 1976.
- *Robert* et *Raymonde Bidochon*, mariés en 1977.

Déboguez ensuite le code suivant :

```
Couple lesClintons;
System.out.println("Bill et Hillary se sont mariés en" +
                   lesClintons.getAnnéeMariage());
```

Couple.java

```
package ex05;

/**
 * Classe représentant un couple marié.
 * @author conujer
 *
 */
public class Couple {
	private String mari;
	private String épouse;
	private int annéeMariage;

	/**
	 * Construit un couple marié.
	 * @param mari                  Prénom et nom du mari.
	 * @param épouse          Prénom et nom de l'épouse.
	 * @param annéeMariage    Année de mariage
	 */
	public Couple(String mari, String épouse, int annéeMariage) {
		this.mari = mari;
		this.épouse = épouse;
		this.annéeMariage = annéeMariage;
```

```
	}

	/**
	 * @return le nom du mari.
	 */
	public String getMari() {
		return mari;
	}

	/**
	 * @return le nom de l'épouse.
	 */
	public String getEpouse() {
		return épouse;
	}

	/**
	 * @return l'année de mariage du couple.
	 */
	public int getAnnéeMariage() {
		return annéeMariage;
	}

	/**
	 * @return une chaîne de caractère décrivant le couple.
	 */
	@Override
	public String toString() {
		return épouse + " et " + mari + " se sont mariés en " + annéeMariage
+ ".";
	}
}
```

Application.java

```
/**
 * Exercices OO : exercice 5.
 * @author conujer
 */
package ex05;

public class Application {

	/**
	 * Point d'entrée du programme.
	 */
	public static void main(String[] args) {
		Couple curies        =    new Couple("Pierre Curie", "Marie Curie",
1895);
		Couple bergerGall = new Couple("Michel Berger", "France Gall", 1976);
		Couple bidochon      =    new  Couple("Robert  Bidochon",  "Raymonde
Bidochon", 1977);

		System.out.println(curies);
		System.out.println(bergerGall);
		System.out.println(bidochon);
	}
}
```

Le code concernant les Clintons crash car l'instance de Couple les représentant n'a pas été instanciée.

3.6 Les calories

L'entreprise qui vous embauche vous demande de créer un programme permettant de compter les calories absorbées en consommant des aliments.

Les aliments contiennent une certaine quantité de lipides, de glucides et de protéines (généralement données en grammes). Ces quantités permettent de savoir combien de calories contient un aliment.

Un compteur de calories se chargera, chaque fois qu'un aliment est mangé, d'accumuler les quantités de lipides, glucides et protéines absorbées et d'en déduire le nombre de calories que cela représente. Si ce nombre dépasse une limite préalablement fixée, un message est affiché.

Plus précisément, un aliment est désigné par un nom et contient un certain nombre de lipides, de glucides et de protéines.

Il faut pouvoir afficher les caractéristiques d'un aliment avec `System.out.println()`. En plus de ces caractéristiques, il faut indiquer si l'aliment en question est sain ou non, au moyen de la méthode de calcul suivante[2] :

*Si nombre lipides [g] * 7 < nombre glucides [g] + nombre protéines : aliment sain.*
Sinon : aliment malsain.

De plus, il faut pouvoir vérifier si deux aliments sont identiques.

Le compteur de calories mémorise la quantité totale de lipides, glucides et protéines absorbées en mangeant des aliments (soit en fournissant directement l'aliment à manger, soit en fournissant la quantité de lipides, glucides et protéines). Il mémorise également une limite (en kilocalories [kcal]) à ne pas dépasser.

Chaque fois qu'un nouvel aliment est mangé, il faut vérifier si la limite de calories n'a pas été dépassée. Si c'est le cas, un message d'avertissement s'affiche sur la console.

Le tableau suivant permet de calculer les calories :

	lipides	glucides	protéines
Nombre de kcal/g	**9**	**4**	**4**

Il faut pouvoir également remettre les compteurs à zéro (sans toucher à la limite).

Le compteur de calories doit également pouvoir nous renseigner sur le pourcentage de graisses (les lipides) absorbées.

Toutes les classes développées doivent implémenter la méthode `toString()`.

Travail à réaliser :

1. À partir du cahier des charges, établir le diagramme de classes.
2. Implémenter l'application selon votre diagramme de classes.
3. Tester l'application avec les informations suivantes :

	lipides	glucides	protéines

[2] Qui ne se base sur aucune rigueur scientifique et qui n'est donc pas digne de confiance.

Un croissant	**4**	**38**	**5**
Une tasse de lait	**101**	**166**	**23**
Un verre de coca	**0**	**35**	**0**
Une portion de frites	**30**	**78**	**7**
Une côte de porc	**75**	**0**	**39**

Limite journalière (par exemple) : 2700 kcal.

4. Une fois tous les aliments avalés, afficher le pourcentage de graisse que cela représente.

Voici un exemple d'affichage :

```
=> Déjeuner
croissant avalé(e) !
tasse de lait avalé(e) ! Mais ce n'est pas sain !
=> Dîner
verre de coca avalé(e) !
pâtes avalé(e) !
Attention, limite de 2700 kcal dépassée.
côte de porc avalé(e) ! Mais ce n'est pas sain !
Attention, limite de 2700 kcal dépassée.
Pourcentage de graisse : 25%
```

Voir le diagramme de classe pour cet exercice, en fin de document.

Aliment.java

```
/**
 * Classe représentant un aliment.
 * @author conujer
 */
package calocount;

public class Aliment {
	private String nom;
	private int lipides;
	private int glucides;
	private int protéines;

	/**
	 * Construit un aliment.
	 * @param nom             Nom de l'aliment.
	 * @param lipides  Quantité de lipides qu'il contient [g].
	 * @param glucides  Quantité de glucides qu'il contient [g].
	 * @param protéines Quantité de protéines qu'il contient [g].
	 */
	public Aliment(String nom, int lipides, int glucides, int protéines) {
		this.nom = nom;
		this.lipides = lipides;
		this.glucides = glucides;
		this.protéines = protéines;
	}

	/**
	 * @return le nom de cet aliment.
	 */
	public String getNom() {
		return nom;
	}

	/**
	 * @return la quantité de lipides [g] contenue dans cet aliment
	 */
	public int getLipides() {
		return lipides;
	}

	/**
	 * @return la quantité de glucides [g] contenue dans cet aliment
	 */
```

```
	public int getGlucides() {
		return glucides;
	}

	/**
	 * @return la quantité de protéines [g] contenue dans cet aliment
	 */
	public int getProtéines() {
		return protéines;
	}

	/**
	 * @return un booléen indiquant si cet aliment est sain ou non.
	 */
	public boolean estSain() {
		return lipides * 7 < glucides + protéines;
	}

	/**
	 * @return une chaîne de caractères décrivant l'aliment.
	 */
	public String toString() {
		String sain = "est";
		if (!estSain())
			sain = "n'est pas";

		return nom + ", contient " + lipides + "g de lipides, " +
		    glucides + "g de glucides, " + protéines +
		    "g de protéines. Cet aliment " + sain + " sain.";
	}

	/**
	 * Compare cet aliment avec un autre.
	 * @param aliment Autre aliment.
	 * @return un booléen indiquant si les deux aliments sont identiques.
	 */
	public boolean equals(Aliment aliment) {
		return nom.equals(aliment.nom)&&
				lipides == aliment.lipides &&
				glucides == aliment.glucides &&
				protéines == aliment.protéines;
	}
}
```

CaloriesCounter.java

```
/**
 * Compteur de calories.
 * @author conujer
 */
package calocount;

public class CaloriesCounter {
	private int lipides = 0;
	private int glucides = 0;
	private int protéines = 0;

	private int limite;
```

```
	/**
	 * Construit un compteur de calories
	 * @param limite Limite [kcal] des calories.
	 */
	public CaloriesCounter(int limite) {
		this.limite = limite;
	}

	/**
	 * Mange l'aliment donné et vérifie si la limite
	 * est dépassée.
	 * @param aliment Aliment à manger.
	 */
	public void manger(Aliment aliment) {
		System.out.println(aliment.getNom() + " avalé(e) !");
		manger(aliment.getLipides(),
				aliment.getGlucides(),
				aliment.getProtéines());
	}

	/**
	 * Mange l'aliment donné sous forme de lipides,
	 * glucides et protéines et vérifie si la limite
	 * est dépassée.
	 * @param lipides   Quantité de lipides [g].
	 * @param glucides  Quantité de glucides [g].
	 * @param protéines Quantité de protéines [g].
	 */
	public void manger(int lipides, int glucides, int protéines) {
		this.lipides += lipides;
		this.glucides += glucides;
		this.protéines += protéines;

		if (getCalories() > limite) {
			System.out.println("Attention, limite de " + limite + " kcal
dépassée.");
		}
	}

	/**
	 * Calcul de nombre de calories avalées.
	 * @return le nombre de calories [kcal].
	 */
	public int getCalories() {
		final int KCALORIES_LIPIDES = 9;
		final int KCALORIES_GLUCIDES = 4;
		final int KCALORIES_PROTEINES = 4;

		return lipides * KCALORIES_LIPIDES +
				glucides * KCALORIES_GLUCIDES +
				protéines * KCALORIES_PROTEINES;
	}

	/**
	 * @return le pourcentage de graisses avalées.
	 */
	public int getPourcentageLipides() {
```

```
            return lipides * 100 / (lipides+glucides+protéines);
      }

      /**
       * Remet les compteurs à zéro (sauf la limite).
       */
      public void reset() {
            lipides = glucides = protéines = 0;
      }

      /**
       * @return une chaîne de caractères décrivant le
       * nombre de lipides, glucides et protéines avalées.
       */
      public String toString() {
            return lipides + "g de lipides, " + glucides + "g de glucides, " +
protéines + "g de protéines.";
      }
}
```

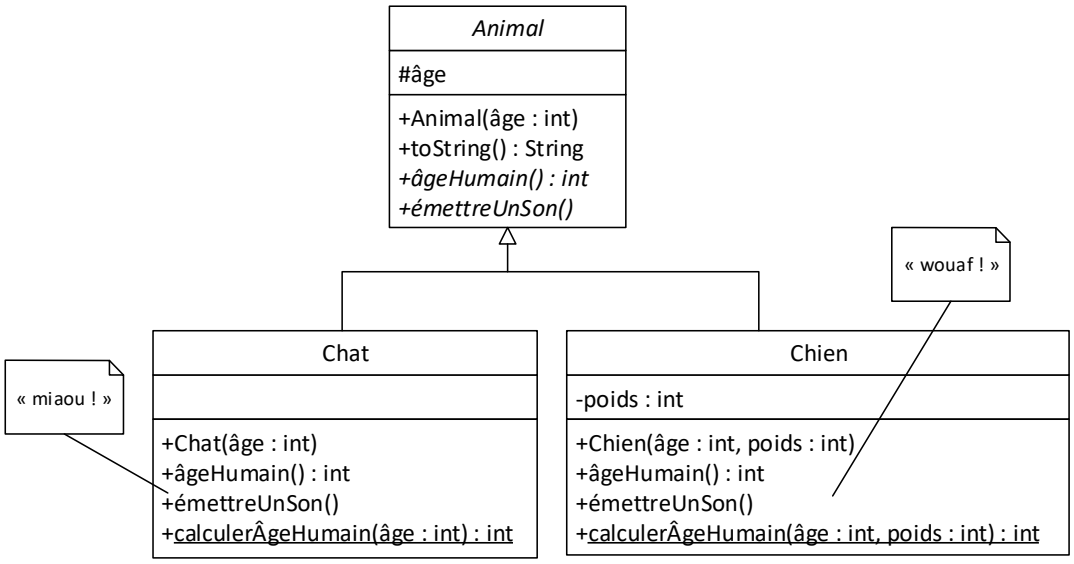
Application.java

```
/**
 * Application qui utilise le compteur de calories
 * @author conujer
 *
 */
package calocount;

public class Application {

      /**
       * Point d'entrée du programme.
       */
      public static void main(String[] args) {
            // 100g de pain      : 0.8g  lip, 54.7g glu, 7g pro
            // 100g de croissant : 5g    lip, 47g   glu, 5g pro
            // 100g de cake      : 18.7g lip, 54.8g glu, 6.4g pro

            // 100g de frite     : 20g   lip, 52g   glu, 5g pro
            // 100g de riz       : 1.7g  lip, 77g   glu, 7.6g pro
            // 100g de pâtes     : 1.4g  lip, 76.5g glu, 12.8g pro
            // 100g de tarte     : 2.8g  lip, 45g   glu, 13g pro

            // 100g de carotte   : 0.3g  lip, 9g    glu, 12g   pro
            // 100g de fraise    : 0.5g  lip, 7g    glu, 0.7g  pro
            // 100g de gruyère   : 29.7g lip, 1.5g  glu, 29g   pro

            // 100g de côte de porc : 30g lip, 0g  glu, 15g pro

            // 100ml coca : 0g lip, 11g glu, 0g pro
            // 100g choc au lait : 33.7g lip, 55.6g glu, 7.6g pro

            final int LIMITE_JOURNALIERE_HOMME = 2700; // kcal

            // Aliments pour le déjeuner
            Aliment croissant = new Aliment("croissant", 4, 38, 5);
            Aliment lait = new Aliment("tasse de lait", 101, 166, 23);
```

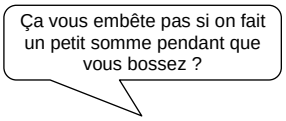
```
            // Aliments pour le repas de midi
            Aliment verreDeCoca = new Aliment("verre de coca", 0, 35, 0); // 300
ml
            Aliment portionFrites = new Aliment("portion de frites", 30, 78, 7);
// 150g
            Aliment côteDePorc = new Aliment("côte de porc", 75, 0, 39); // 250g

            // Utilisation du compteur de calories
            CaloriesCounter             compteur             =             new
CaloriesCounter(LIMITE_JOURNALIERE_HOMME);

            System.out.println("=> Déjeuner");
            compteur.manger(croissant);
            compteur.manger(lait);

            System.out.println("=> Dîner");
            compteur.manger(verreDeCoca);
            compteur.manger(portionFrites);
            compteur.manger(côteDePorc);

            System.out.println("Pourcentage     de     graisse     :     "     +
compteur.getPourcentageLipides() + "%");

      }

}
```

3.7 Chien et chat

Implémenter en *Java* le diagramme de classes suivant :



Créer une application qui instancie les animaux suivants, leur fait émettre un son, puis les affiche sur la console :



médor:Chien
âge : 7 ans Poids : 15 kilos

minet:Chat
âge : 5 ans

Voici les formules[3] permettant d'obtenir l'âge humain équivalent pour les chiens et les chats :

Chien de moins de 15 kilos	$f(\hat{a}ge) = \begin{cases} 20, & \hat{a}ge = 1 \\ 28 + (\hat{a}ge - 2) \cdot 4, & \hat{a}ge \geq 2 \end{cases}$
Chien de 15 kilos à 40 kilos.	$f(\hat{a}ge) = \begin{cases} 18, & \hat{a}ge = 1 \\ 27 + (\hat{a}ge - 2) \cdot 6, & \hat{a}ge \geq 2 \end{cases}$
Chien de plus de 40 kilos.	$f(\hat{a}ge) = \begin{cases} 16, & \hat{a}ge = 1 \\ 22 + (\hat{a}ge - 2) \cdot 9, & \hat{a}ge \geq 2 \end{cases}$
Chat	$f(\hat{a}ge) = \begin{cases} 19, & \hat{a}ge = 1 \\ 24 + (\hat{a}ge - 2) \cdot 4, & \hat{a}ge \geq 2 \end{cases}$

Voici un exemple d'affichage :

```
wouaf !
miaou !
J'ai 7 ans (57 ans humain).
J'ai 5 ans (36 ans humain).
```

Animal.java

```
/**
 * Classe représentant un animal.
 */
package ex07;

public abstract class Animal {
	protected int âge;

	/**
	 * Construit un animal d'un certain âge.
	 * @param âge Âge de l'animal.
	 */
	public Animal(int âge) {
		this.âge = âge;
	}

	/**
	 * @return une chaîne indiquant l'âge de l'animal et son âge humain.
	 */
	public String toString() {
		return "J'ai " + âge + " ans (" + getÂgeHumain() + " ans humain).";
    }

	/**
```

[3] Chiffres tirés de http://www.age-humain.com/

```
        * Calcul l'âge équivalent humain de cet animal.
        * @return l'âge équivalent humain.
        */
       public abstract int getÂgeHumain();

       /**
        * Emet un son.
        */
       public abstract void emettreUnSon();
}
```

Chat.java

```
/**
 * Classe représentant un chat.
 */
package ex07;

public class Chat extends Animal {

       /**
        * Construit un chat d'un certain âge.
        * @param âge Âge du chat.
        */
       public Chat(int âge) {
              super(âge);
       }

       /**
        * Calcul l'âge équivalent humain de ce chat.
        * @return l'âge équivalent humain.
        */
    public int getÂgeHumain() {
        return calculerÂgeHumain(âge);
    }

       /**
        * Emet un son.
        */
       public void emettreUnSon() {
              System.out.println("miaou !");
       }

       /**
        * Calcule l'âge équivalent humain d'un chat.
        * @param âge      Âge du chat.
        * @return l'âge équivalent humain du chat.
        */
    public static int calculerÂgeHumain(int âge) {
       if (âge < 1)
           return 0;

       if (âge == 1)
           return 19;

       return 24 + (âge-2) * 4;
    }
}
```

Chien.java

```
/**
 * Classe représentant un chien.
 */
package ex07;

public class Chien extends Animal {

	private int poids;

	/**
	 * Construit un chien d'un certain âge.
	 * @param âge    Âge du chien.
	 * @param poids  Poids du chien [kg].
	 */
	public Chien(int âge, int poids) {
		super(âge);
		this.poids = poids;
	}

	/**
	 * Calcul l'âge équivalent humain de ce chien.
	 * @return l'âge équivalent humain.
	 */
	public int getÂgeHumain() {
	    return calculerÂgeHumain(âge, poids);
	}

	/**
	 * Emet un son.
	 */
	public void emettreUnSon() {
		System.out.println("wouaf !");
	}

	/**
	 * Calcule l'âge équivalent humain d'un chien.
	 * @param âge      Âge du chien.
	 * @param poids Poids du chien.
	 * @return l'âge équivalent humain du chien.
	 */
	public static int calculerÂgeHumain(int âge, int poids) {
	    if (âge < 1)
	        return 0;

	    if (poids < 15) {
	        if (âge == 1)
	            return 20;
	        else
	            return 28 + (âge-2)*4;
	    } else if (poids > 40) {
	        if (âge == 1)
	            return 16;
	        else
	            return 22 + (âge-2)*9;
	    } else {
	        if (âge == 1)
```

```
                    return 18;
                else
                    return 27 + (âge-2) * 6;
            }
}

}
```

Application.java

```
/**
 * Application qui teste l'exercice.
 * @author conujer
 */

package ex07;

public class Application {

	/**
	 * Point d'entrée du programme.
	 */
	public static void main(String[] args) {
		Chat  minet = new Chat(5);
		Chien médor = new Chien(7, 15);

		minet.emettreUnSon();
		médor.emettreUnSon();
	}

}
```

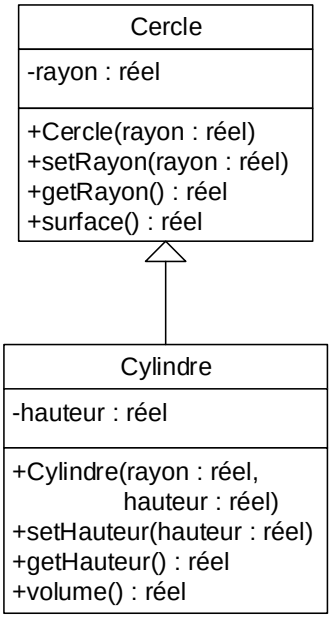
3.8 Cercle et cylindre

Une application qui gère quelques formes géométriques 2D possède la classe suivante :

Cercle
-rayon : réel
+Cercle(rayon : réel) +setRayon(rayon : réel) +getRayon() : réel +surface() : réel

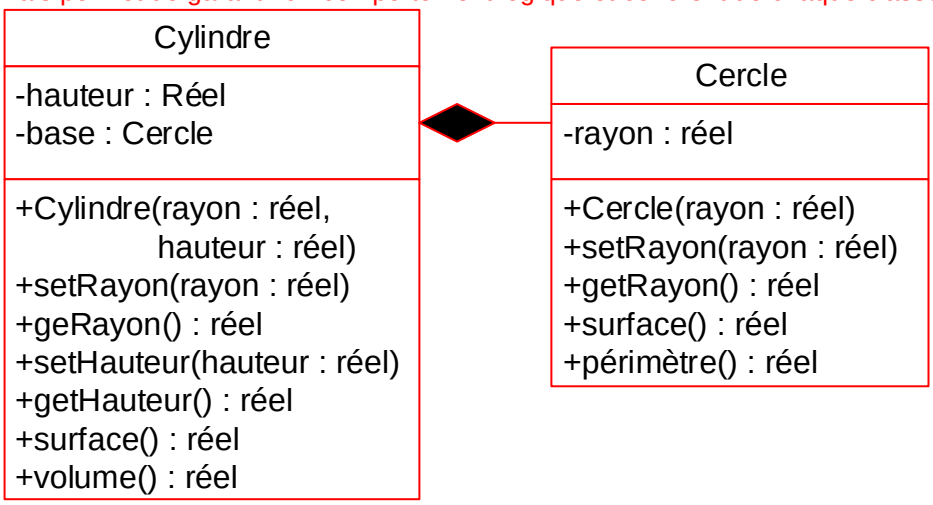
Vous êtes chargé de créer une classe représentant un cylindre et permettant de définir la hauteur du cylindre et de calculer son volume.

Votre chef vous propose d'implémenter le diagramme de classes suivant :



Que pensez-vous de sa suggestion ? Quels sont les avantages et inconvénients de cette solution ? Y aurait-il une meilleure solution ?

Considérer que le cylindre EST UN cercle provient d'une spécificité mathématique (ou géométrique) partant de l'idée que le cercle est le cas particulier d'un cylindre de hauteur nulle.
Cette solution présente également l'avantage de ne pas avoir à réécrire le code gérant le rayon du cylindre.
Toutefois, un problème se pose avec la méthode surface : si j'instancie un Cylindre et que j'appelle sa méthode surface, cela me retourne la surface de la base (du cercle) et non pas du cylindre.
Pour corriger cela, il faudrait réimplémenter la méthode surface() dans la classe Cylindre. Mais dès lors, je ne peux plus manipuler un cercle sans me soucier de son type exact : le calcul de la surface par le cylindre n'est plus correcte du point de vue du cercle, ce qui pose problème.
Autre exemple : que faire si la classe Cercle ajoute une méthode périmètre ? Qu'est-ce que la notion de périmètre dans le cas d'un cylindre ?
La meilleure solution serait d'utiliser le cercle comme une composition. Cela nécessite un peu plus de code, mais permet de garantir un comportement logique et cohérent de chaque classe.



À noter que pour résoudre ce genre de cas, tout en bénéficiant de la récupération du code de la classe Cercle, certains langages permettent l'héritage **privé**.

3.9 Ulysse 31

Vous êtes chargé par une entreprise informatique de participer à la conception d'un jeu vidéo basé sur l'univers d'une vieille série animée dénommée *Ulysse 31*.

Plus particulièrement, vous êtes en charge de modéliser tout ce qui concerne l'équipage d'un vaisseau et ses membres. Voici les informations vous permettant d'y arriver :

Un vaisseau est caractérisé par un nom et contient un équipage.

Les membres de l'équipage peuvent être de différentes races : les humains, les zotriens et les robots.

Ils ont tous un nom et un âge. Un membre de l'équipage est capable de dire si oui ou non il sait piloter un vaisseau.

Un humain n'a pas de pouvoir particulier. Un zotrien est capable de lire dans les pensées et un robot est capable de calculer.

Un robot possède une caractéristique particulière : sa vitesse de processeur (un nombre réel).

Selon leur race et leurs caractéristiques, les membres de l'équipage sont capables de faire preuve de plus ou moins de sagesse. Cela est représenté par des points de sagesse (un nombre entier), calculés ainsi :

Race	**Points de sagesse**
Humain	Si âge < 12 : zéro Si âge ≥ 90 : zéro Sinon : âge + 20
Zotrien	√âge
Robot	(Vitesse du processeur)$^{\pi}$

Il est possible de connaître le membre de l'équipage qui fait preuve de moins de sagesse (si plusieurs membres ont la même sagesse, renvoyer le dernier de la liste).

Lorsqu'un équipage est affecté à un vaisseau, le premier membre sachant piloter est désigné comme pilote du vaisseau (si au sein de l'équipage, personne ne sait piloter, le vaisseau refuse d'utiliser l'équipage proposé).

Un vaisseau doit pouvoir indiquer qui est son pilote.

La méthode `toString()` du vaisseau doit retourner une chaîne ayant le format suivant :

```
Nom vaisseau : taille équipage membres d'équipage. Pilote : nom pilote
```

Travail à réaliser :

1. À partir du cahier des charges, établir le diagramme de classes.
2. Implémenter l'application selon votre diagramme de classes.
3. Tester l'application avec les informations suivantes :

   Nom du vaisseau : Odysseus

   Membres de l'équipage :

   Ulysse, 40 ans, humain, sait piloter.
   Télémaque, 13 ans, humain, ne sait pas piloter.
   Thémis, 11 ans, zotrienne, ne sait pas piloter.
   Nono, 1 ans, robot, sait piloter, vitesse du processeur : 3.44.

Voici un exemple d'affichage :

```
Odysseus : 4 membres d'équipage. Pilote : Ulysse
Le membre d'équipage ayant le moins de sagesse est Thémis.
```

Pour le diagramme de classe, voir indice page suivante.
Créer un projet Ulysse31, un package equipage (qui contient également le vaisseau) et un package application.

MembreEquipage.java

```
/**
 * Classe représentant un membre d'équipage.
 * @author conujer
 */
package equipage;

public abstract class MembreEquipage {

	private String nom;
	protected int âge;
	private boolean saitPiloter;

	/**
	 * Construit un membre d'équipage.
	 * @param nom                    Nom du membre.
	 * @param âge                    Âge du membre.
	 * @param saitPiloter   Indique s'il sait piloter.
	 */
	public MembreEquipage(String nom, int âge, boolean saitPiloter) {
		this.nom = nom;
		setAge(âge);
		this.saitPiloter = saitPiloter;
	}

	/**
	 * @return le nom de ce membre.
	 */
	public String getNom() {
		return nom;
	}

	/**
	 * @return l'âge de ce membre.
	 */
	public int getAge() {
		return âge;
	}

	/**
	 * @return un booléen indiquant si ce membre sait piloter.
	 */
	public boolean saitPiloter() {
		return saitPiloter;
	}

	/**
	 * Change l'âge de ce membre.
	 * @param âge Nouvel âge.
	 */
	private void setAge(int âge) {
```

```
            if (âge < 0)
                  âge = 0;
            this.âge = âge;
      }

      /**
       * @return la sagesse de ce membre.
       */
      public abstract int getSagesse();

}
```

Humain.java

```
/**
 * Classe représentant un membre d'équipage humain.
 * @author conujer
 */
package equipage;

public class Humain extends MembreEquipage {

      /**
       * Construit un humain.
       * @param nom                  Nom de l'humain.
       * @param âge                  Âge de l'humain.
       * @param saitPiloter   Indique si l'humain sait piloter.
       */
      public Humain(String nom, int âge, boolean saitPiloter) {
            super(nom, âge, saitPiloter);
      }

      /**
       * @return la sagesse de cet humain.
       */
      public int getSagesse() {
            if ((âge < 12) || (âge >= 90))
                  return 0;
            else
                  return âge + 20;
      }
}
```

Zotrien.java

```
/**
 * Classe représentant un membre d'équipage zotrien.
 * @author conujer
 */
package equipage;

public class Zotrien extends MembreEquipage {

      /**
       * Construit un zotrien.
       * @param nom                  Nom du zotrien.
       * @param âge                  Âge du zotrien.
       * @param saitPiloter   Indique s'il sait piloter.
       */
      public Zotrien(String nom, int âge, boolean saitPiloter) {
            super(nom, âge, saitPiloter);
```

```
	}

	/**
	 * @return la sagesse de ce zotrien.
	 */
	public int getSagesse() {
		return (int)Math.sqrt(âge);
	}

	/**
	 * Lit dans les pensées de quelqu'un.
	 */
	public void lireLesPensées() {
		System.out.println("Je lis les pensées.");
	}

}
```

Robot.java

```
/**
 * Classe représentant un membre d'équipage robot.
 * @author conujer
 */
package equipage;

public class Robot extends MembreEquipage{

	private double vitesseProcesseur;

	/**
	 * Construit un robot.
	 * @param nom                    Nom du robot.
	 * @param âge                    Âge du robot.
	 * @param saitPiloter   Indique s'il sait piloter.
	 */
	public   Robot(String   nom,   int   âge,   boolean   saitPiloter,   double
vitesseProcesseur) {
		super(nom, âge, saitPiloter);
		this.vitesseProcesseur = vitesseProcesseur;
	}

	/**
	 * @return la sagesse de ce robot.
	 */
	public int getSagesse() {
		return (int)Math.pow(vitesseProcesseur, Math.PI);
	}

	/**
	 * Calcule quelque chose.
	 */
	public void calculer() {
		System.out.println("Je calcule.");
	}
}
```

Equipage.java

```
/**
 * Classe représentant un équipage.
```

```
 * @author conujer
 */
package equipage;

import java.util.ArrayList;

public class Equipage {
	private List<MembreEquipage> équipage = new ArrayList<>();

	/**
	 * Ajoute un membre à l'équipage.
	 * @param membre Membre à ajouter.
	 */
	public void ajouterMembre(MembreEquipage membre) {
		if (!équipage.contains(membre))
			équipage.add(membre);
	}

	/**
	 * Recherche le membre de l'équipage le moins sage.
	 * Si plusieurs membres sont les moins sages, c'est le dernier qui est retenu.
	 * @return le membre de l'équipage le moins sage.
	 */
	public MembreEquipage getMembreLeMoinsSage() {
		MembreEquipage leMoinsSage = null;
		int sagesseMinimale = Integer.MAX_VALUE;

		for(MembreEquipage membre : équipage) {
			if (membre.getSagesse() <= sagesseMinimale) {
				leMoinsSage = membre;
				sagesseMinimale = membre.getSagesse();
			}
		}
		return leMoinsSage;
	}

	/**
	 * @return le nombre de membres composant l'équipage.
	 */
	public int getTailleEquipage() {
		return équipage.size();
	}

	/**
	 * @return le nombre de membres de l'équipage sachant piloter.
	 */
	public int getNombrePilotes() {
		return getPilotes().size();
	}

	/**
	 * @return la liste des membres de l'équipage sachant piloter.
	 */
	public List<MembreEquipage> getPilotes() {

		List<MembreEquipage> pilotes = new ArrayList<>();

		for(MembreEquipage membre : équipage) {
```

```
                    if (membre.saitPiloter()) {
                        pilotes.add(membre);
                    }
                }
                return pilotes;
        }
}
```

Vaisseau.java

```
/**
 * Classe représentant un vaisseau.
 * @author conujer
 */
package equipage;

public class Vaisseau {
        private String nom;
        private Equipage équipage = null;

        /**
         * Construit un vaisseau.
         * @param nom  Nom du vaisseau.
         */
        public Vaisseau(String nom) {
                this.nom = nom;
        }

        /**
         * Affecte un équipage à ce vaisseau.
         * L'équipage doit posséder au moins un pilote,
         * sinon il n'est pas pris en compte.
         * @param équipage  Nouvel équipage du vaisseau.
         */
        public void setEquipage(Equipage équipage) {
                if (équipage.getNombrePilotes() > 0)
                        this.équipage = équipage;
                else
                        System.out.println("L'équipage doit au moins posséder un
pilote.");
        }

        /**
         * Indique qui est le pilote du vaisseau.
         * Le pilote du vaisseau est le premier membre de l'équipage sachant piloter.
         * @return le membre d'équipage qui pilote ce vaisseau.
         */
        public MembreEquipage getPilote() {
                if (équipage == null) {
                        System.out.println("Le vaisseau ne possède aucun équipage.");
                        return null;
                }
                return équipage.getPilotes().get(0);
        }

        /**
         * @return une description de ce vaisseau.
         */
        public String toString() {
```

```
            if (équipage == null)
                  return nom + " : ne possède pas d'équipage.";
            return nom + " : " + équipage.getTailleEquipage() + " membres
d'équipage. Pilote : " + getPilote().getNom();
      }
}
```

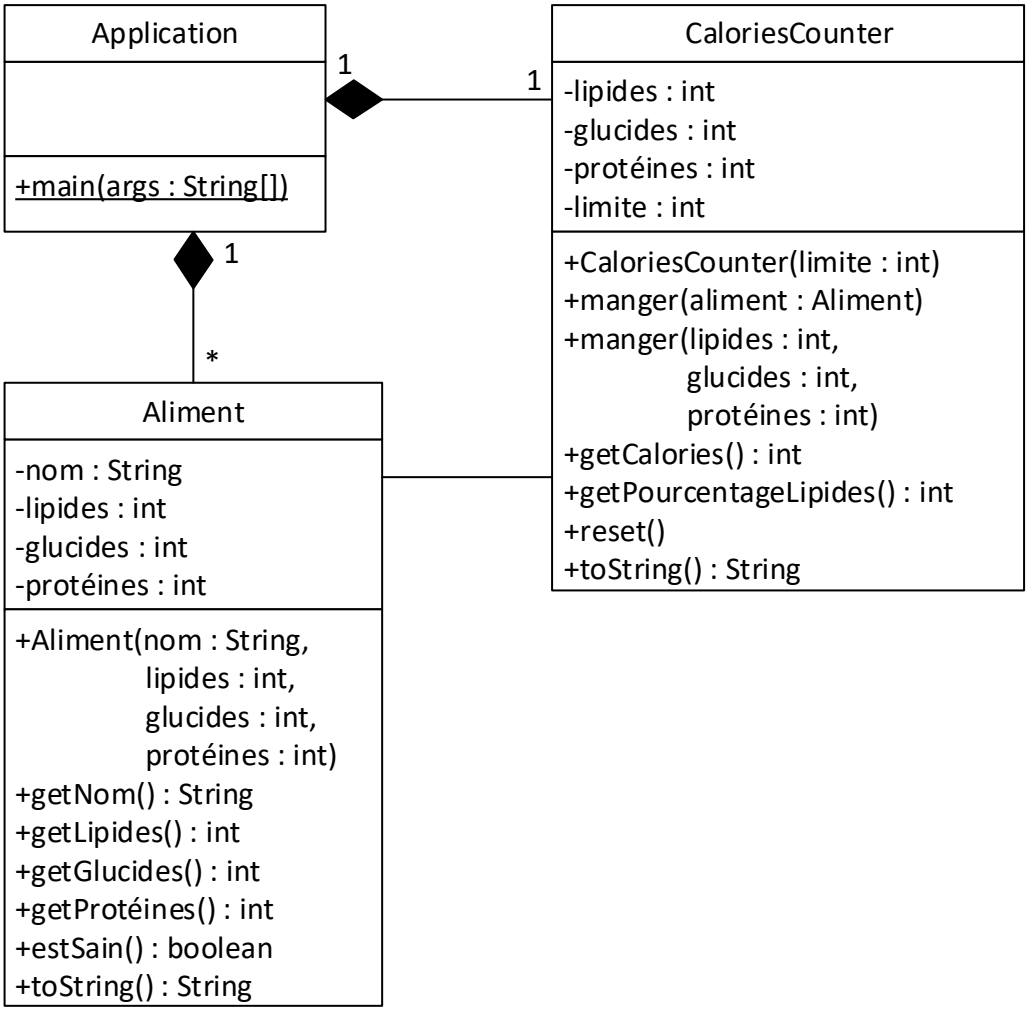
Application.java

```
/**
 * Programme de test de l'exercice Ulysse 31.
 */
package application;

public class Application {

      /**
       * Point d'entrée du programme.
       */
      public static void main(String[] args) {
            Vaisseau vaisseau = new Vaisseau("Odysseus");
            Equipage équipage = new Equipage();

            équipage.ajouterMembre(new Humain("Ulysse", 40, true));
            équipage.ajouterMembre(new Humain("Télémaque", 13, false));
            équipage.ajouterMembre(new Zotrien("Thémis", 11, false));
            équipage.ajouterMembre(new Robot("Nono", 1, true, 3.44));

            vaisseau.setEquipage(équipage);

            System.out.println(vaisseau);
            System.out.println("Le membre d'équipage ayant le moins de sagesse est
" +
                              équipage.getMembreLeMoinsSage().getNom() + ".");

      }
}
```

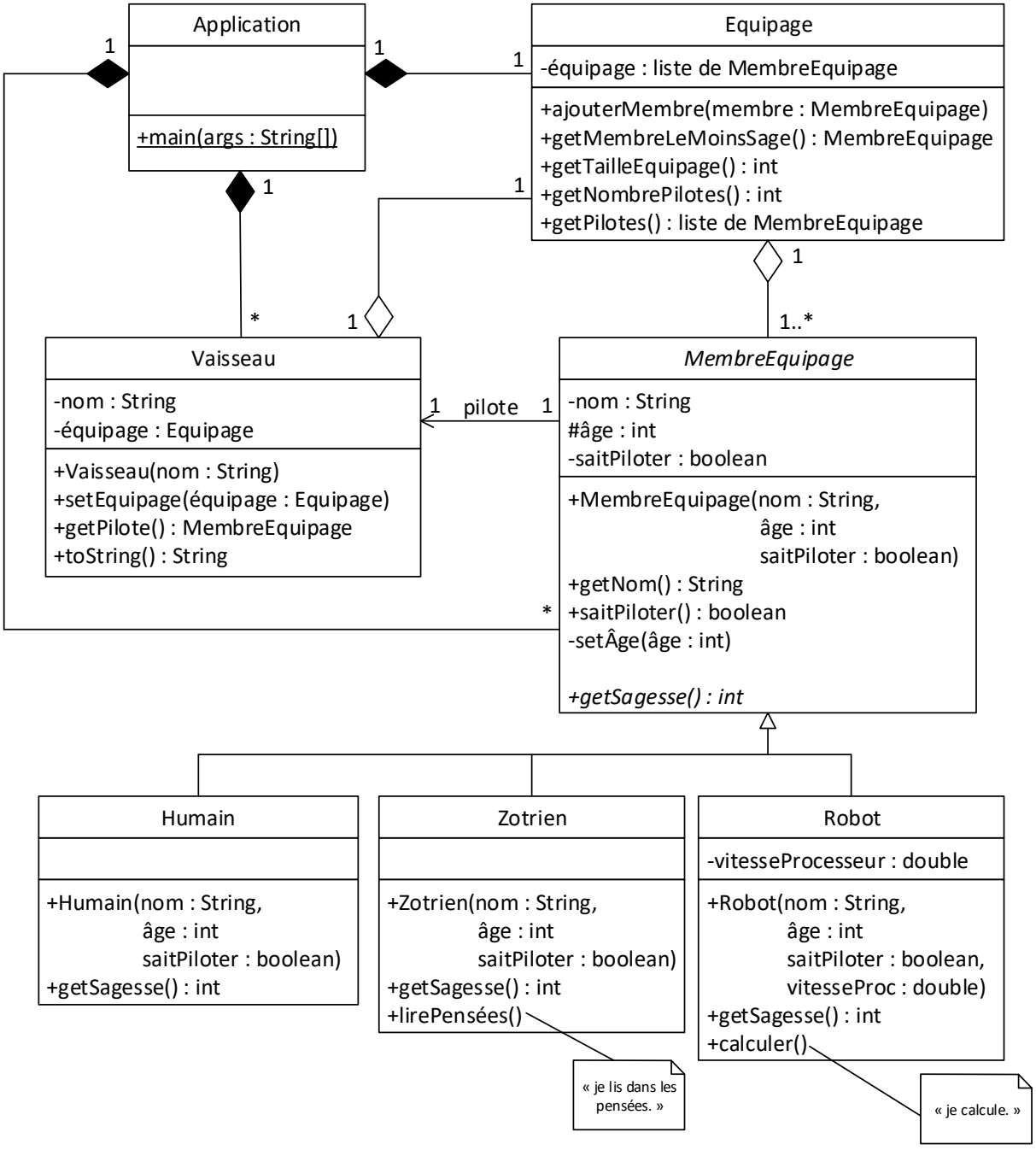
4 Aides

4.1 Les calories

Voici un diagramme de classe envisageable :



4.2 Ulysse 31


Application
+main(args : String[])
Equipage
-équipage : liste de MembreEquipage
+ajouterMembre(membre : MembreEquipage)
+getMembreLeMoinsSage() : MembreEquipage
+getTailleEquipage() : int
+getNombrePilotes() : int
+getPilotes() : liste de MembreEquipage
Vaisseau
-nom : String
-équipage : Equipage
+Vaisseau(nom : String)
+setEquipage(équipage : Equipage)
+getPilote() : MembreEquipage
+toString() : String
pilote
MembreEquipage
-nom : String
#âge : int
-saitPiloter : boolean
+MembreEquipage(nom : String, âge : int, saitPiloter : boolean)
+getNom() : String
+saitPiloter() : boolean
-setÂge(âge : int)
+getSagesse() : int
Humain
+Humain(nom : String, âge : int, saitPiloter : boolean)
+getSagesse() : int
Zotrien
+Zotrien(nom : String, âge : int, saitPiloter : boolean)
+getSagesse() : int
+lirePensées()
« je lis dans les pensées. »
Robot
-vitesseProcesseur : double
+Robot(nom : String, âge : int, saitPiloter : boolean, vitesseProc : double)
+getSagesse() : int
+calculer()
« je calcule. »