Université de Lille

SAÉ S2.01-02

Programmation orientée objet

Rapport

Conception & Développement

Élèves :
Baptiste Lavogiez
Mark Zavadskiyi
Angèl Zheng

Enseignant :
Antoine Nongaillard
Fabien Delecroix

20 mai 2025

Table des matières

1 Rendu 1 **3**
1.1 Réflexion . . . 3
1.2 UML . . . 3
1.3 Remarques . . . 3
1.4 Implémentation . . . 4

2 Rendu 2 **7**
2.1 Attentes . . . 7
2.2 Réflexion . . . 7
2.3 UML . . . 7
2.4 Modifications de `Person` . . . 8
2.5 Modifications de `Criteria` . . . 10
2.6 Modifications de `Exchange` . . . 13
2.7 Classes de Test . . . 14

3 Rendu 3 **16**

4 Rendu 4 **17**

Ce rapport présentera la partie **Conception & Développement** de la SAÉ S2.01-02. Elle traitera d'une solution à la situation d'échange scolaire entre deux personnes régie par des critères et exigences formulées ou non.

1 Rendu 1

Ce premier rendu traitera de la conception des classes des bases, des contraintes et des classes de tests.

1.1 Réflexion

Cette semaine, nous avons à poser les bases de notre projet. Nous réflechissons d'abord aux structures les plus adaptées pour le besoin, qui est ici d'échanger des élèves entre eux sur la base de critères.

Nous avons besoin d'une Personne, de Critères stockés avec leur types, et d'un Echange entre deux Personnes. Cela se matérialise avec un UML.

1.2 UML

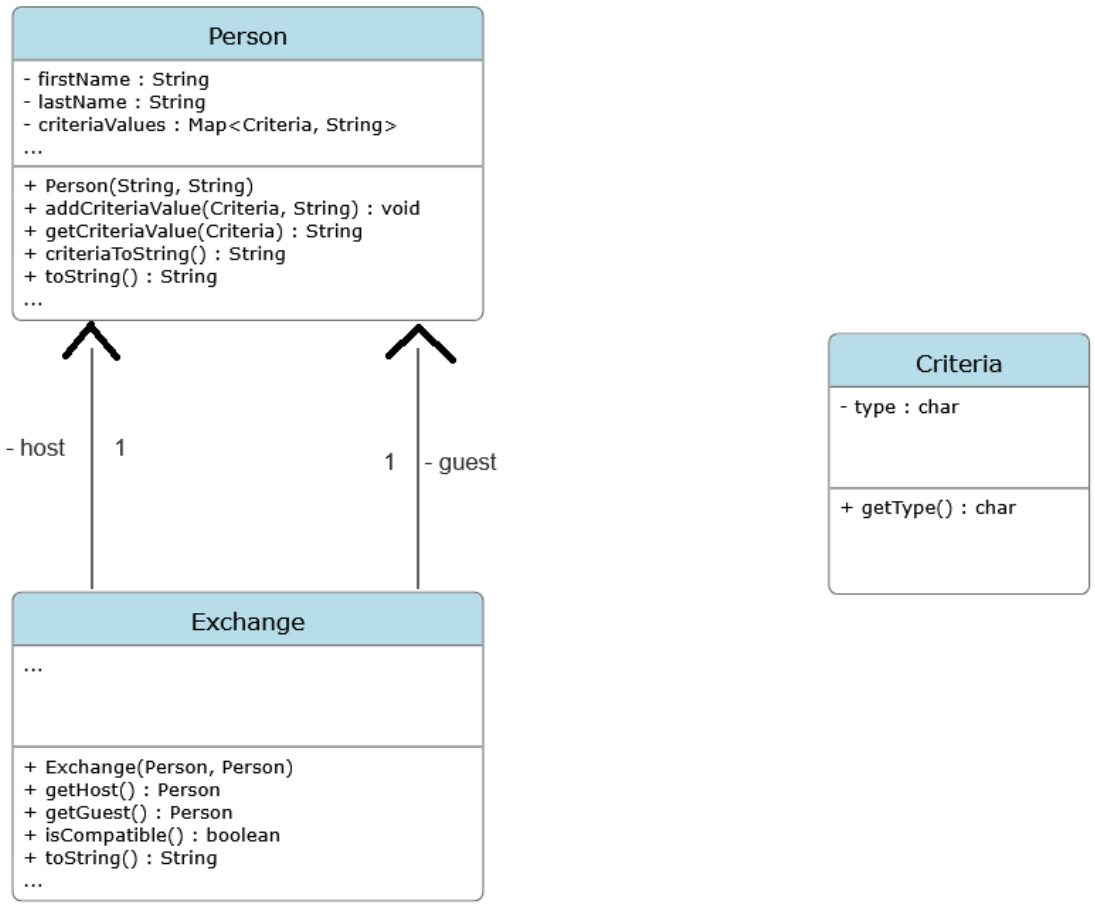

FIGURE 1 – Relations UML

1.3 Remarques

Nous avons décidé d'une représentation minimaliste mais avec tous les éléments nécessaires. Le code est facilement compréhensible avec des noms explicites.

1. Des critères...

Criteria est une énumération simple de critères associés à un type représenté sous forme de caractère.

Cette énumération est utilisée dans criteriaValues, la Map de la classe Person. Elle y sera associée à une valeur représentée en chaîne de caractères pour simplifier la modélisation. Dans Person, les méthodes adéquates permettent de manipuler ces critères (ajout, obtention, conversion en texte...).

2. Person ou Student ?

Ici, le fait que les personnes soient des étudiants n'est pas important. Choisir le modèle d'un étudiant reviendrait à devoir encapsuler une personne dans la classe, soit une classe de plus.

3. Le double rôle

Une personne peut aussi bien être un hote qu'un guest. Pour cela, Exchange encapsule les deux afin d'avoir une combinaison claire et simple à manipuler.

4. Critiques

La classe Exchange aurait pu, de façon plus ou moins simple, se raccourcir en étant définie dans Person. Le problème étant que de cette façon, le rôle de Host et de Guest devrait être défini autrement. Nous préférons Exchange pour fonctionner de manière plus simple afin de manipuler clairement les rôles.

1.4 Implémentation

L'implémentation se fait de manière naturelle après avoir tout défini de façon claire. Des tests élargis sont disponibles dans le dossier sous-jaçent et permettent de voir comment le modèle réagit à toutes les situations.

1. Aperçu de Exchange

Java

```java
package v1;

public class Exchange {
    private Person host;
    private Person guest;

    public Exchange(Person host, Person guest) {
        this.host = host;
        this.guest = guest;
    }

    public Person getHost() {
        return this.host;
    }

    public Person getGuest() {
        return this.guest;
    }
```

2. Aperçu de ExchangeTest

Java

```java
package v1 ;

public class ExchangeTest {

    private Person alice;
    private Person bob;
    private Exchange exchange;

    @BeforeEach
    public void initTest() {
        alice = new Person("Astana", "Nur-sultan");
        bob = new Person("Bob", "Marley");
        exchange = new Exchange(alice, bob);
    }

    @Test
    public void testIncompatibleDueToAnimalAllergy() {
        alice.addCriteriaValue(Criteria.HOST_HAS_ANIMAL, "true");
        bob.addCriteriaValue(Criteria.GUEST_ANIMAL_ALLERGY, "true");

        assertFalse(exchange.isCompatible());
    }

    @Test
    public void testIncompatibleDueToGenderPreference() {
        alice.addCriteriaValue(Criteria.HOST_HAS_ANIMAL, "false");
        bob.addCriteriaValue(Criteria.GUEST_ANIMAL_ALLERGY, "false");
        alice.addCriteriaValue(Criteria.PREFERENCE_GENDER, "Female");
        bob.addCriteriaValue(Criteria.GENDER, "Male");

        assertFalse(exchange.isCompatible());
    }
```

2 Rendu 2

2.1 Attentes

Cette semaine, nous traiterons de :

- Gestion de la validité des critères par un mécanisme d'exception
- Développement des règles spécifiques de compatibilité pour certains pays

La règle spécifique de compatibilité étant, pour le moment, celle énonçant qu'un échange comprenant une personen française devait avoir au moins un hobby en commun sans quoi le risque de cassure serait trop important.
La validité des critères portera sur :

- La bonne entrée d'un type (car jusqu'ici ils sont tous en String) ; exemple : HOST_HAS_ANIMAL doit être à true ou false.
- La cohérence des valeurs entrées. Par exemple une Person ne peut pas être allergique aux animaux et en avoir un.

2.2 Réflexion

Nous avons réfléchi à comment implémenter les critères spécifiques aux pays.
Nous avons pensé à :

- Une énumération `SpecificCountryRule`
- Une ArrayList de "rules" au sein de `Person`

Au final, nous avons opté pour une définition dans `Person` au sein de la liste de critères.
La validité des critères, elle, a été validée en divisant la vérification en deux temps : d'abord le bon respect du type prévu (pas de `yes` dans un boolean), pour ensuite vérifier la bonne valeur (pas de `toto` dans la date de naissance).

2.3 UML

L'UML est, par conséquent, mis à jour et amélioré dans ses détails.

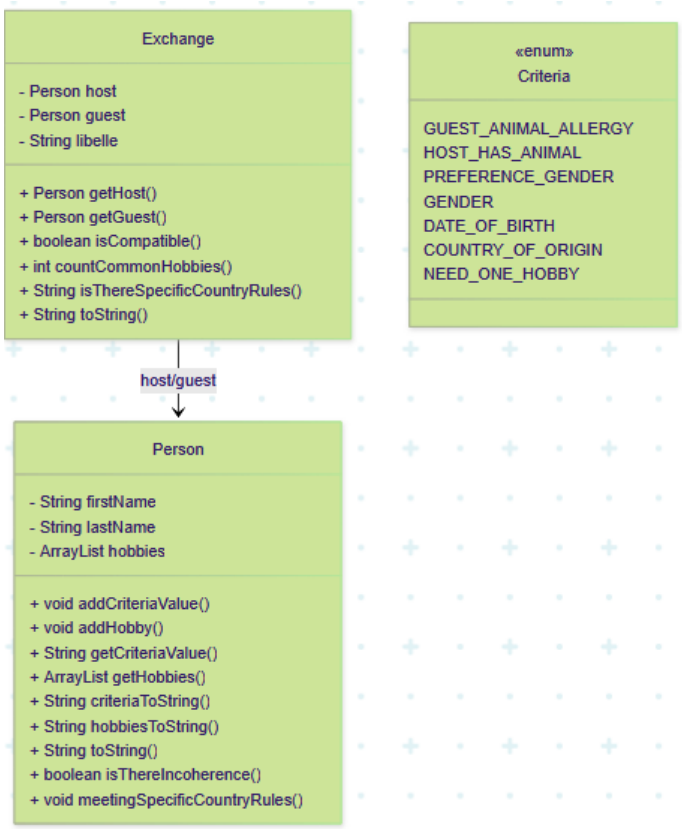

FIGURE 2 – Relations UML

2.4 Modifications de `Person`

Pour ce faire, nous avons procédé à l'implémentation d'une méthode `meetingSpecificCountryRules` dans la classe `Person`.

Elle est appelée dans le constructeur désormais surchargé et à l'ajout d'un nouveau critère. En somme, elle est appelée à chaque possible changement de critère pour adapter la bonne logique.

Java

```java
    // Cette méthode change les critères en fonction du pays
        // Elle doit être appelée à chaque fois qu'on change un critère (constructeur, ajout)
public void meetingSpecificCountryRules() {
        try {
                String country = this.getCriteriaValue(Criteria.COUNTRY_OF_ORIGIN).toUpperCase();
                if (country.equals("FRANCE")) {
                        this.addCriteriaValue(Criteria.NEED_ONE_HOBBY, "true");
                }
        } catch (NullPointerException e) {
                // Si le pays n'est pas défini, on ne fait rien, ce n'est pas vraiment une erreur
        }
}
```

Dans la même idée et dans la même classe, une méthode `isThereIncoherence` a été ajoutée. Elle veille à faire respecter les contradictions de données ; ici, il est sujet d'une personne allergique aux animaux mais en ayant un aussi en même temps.

Java

```java
public boolean isThereIncoherence() {
        // On vérifie si la personne a un animal et est allergique
        String hostHasAnimal = this.getCriteriaValue(Criteria.HOST_HAS_ANIMAL);
        String guestAllergy = this.getCriteriaValue(Criteria.GUEST_ANIMAL_ALLERGY);

        if ("true".equals(guestAllergy) && "true".equals(hostHasAnimal)) {
                return true;
        }
        return false;
}
```

De plus, une `ArrayList` de `hobbies` a été ajoutée, en accord avec les critères relatifs à un nombre de hobbies en commun minimal.

2.5 Modifications de `Criteria`

Un critère a été ajouté : `NEED_ONE_HOBBY`, caractérisant le besoin dans un `Exchange` d'avoir au moins un hobby en commun en cas de règles spécifiques. Ce critère est un booléen ('B').

La vérification des critères se décompose en deux parties : type, puis valeur.

1. La méthode `static isCriteriaTypeValid`

Java

```java
public static boolean isCriteriaTypeValid(Criteria criteria,
    String value) {
        char type = criteria.getType();
        switch(type) {
                case 'B':
                value = value.toLowerCase(); // on met tout
                    en minuscule
                // On vérifie si la valeur est "true" ou "
                    false"
                return value.equals("true") || value.equals(
                    "false");
                case 'T':
                return (value.length()>0); // TODO: Add
                    validation for text criteria
                case 'N':
                try {
                        Integer.parseInt(value);
                        return true;
                } catch (NumberFormatException e) {
                        System.out.println("Invalid number
                            format: " + value);
                        System.out.println("Exception: " + e
                            .getMessage());
                        return false;
                }
                case 'D':
        }
        return false;
}
```

Cette méthode sera appelée, en complément de la suivante, à chaque nouvel ajout de critère. Si les méthodes ne sont pas satisfaites, l'ajout échoue.

2. La méthode `static isCriteriaValueValid`

Java

```java
public static boolean isCriteriaValueValid(Criteria criteria, String value) {
        value = value.toLowerCase(); // on met tout en minuscule
        // Mode en if car pas toutes les valeurs sont concernées
        if(criteria==Criteria.PREFERENCE_GENDER || criteria==Criteria.GENDER) return value.equals("male") || value.equals("female") || value.equals("other");
        if(criteria==Criteria.DATE_OF_BIRTH) {
                try {   //Cette partie gestion d'excception est plutôt à mettre lors de l'entrée des données.
                        return LocalDate.now().minusYears(18).isAfter((LocalDate.parse(value)));
                } catch (DateTimeParseException e) {
                        System.out.println("Format invalide, utilisez le format suivant : yyyy-MM-dd.");
                        return false;
                }
        }
        return true ;
}
```

3. La méthode `static areCriteriasValid`

Java

```java
    // Vérifie l'ensemble des critères et leur validité. Si un n'est pas valide, on renvoie false.
public static boolean areCriteriasValid(Map<Criteria, String> criterias) {
        for (Map.Entry<Criteria, String> entry : criterias.entrySet()) {
            Criteria criteria = entry.getKey();
            String value = entry.getValue();
            try {
                if (!isCriteriaTypeValid(criteria, value)) {
                    System.out.println("Invalid type for criteria " + criteria + ": " + value);
                    return false;
                }
            } catch (Exception e) {
                System.out.println("Exception during type validation for " + criteria + ": " + e.getMessage());
                return false;
            }
            try {
                if (!isCriteriaValueValid(criteria, value)) {
                    System.out.println("Invalid value for criteria " + criteria + ": " + value);
                    return false;
                }
            } catch (Exception e) {
                System.out.println("Exception during value validation for " + criteria + ": " + e.getMessage());
                return false;
            }
        }
        return true;
}
```

Cette méthode est appelée dans le constructeur prenant en entrée une la `Map` des critères de la `Person`.

2.6 Modifications de `Exchange`

1. La méthode `countCommonHobbies`

Java

```java
//Donne le nombre de centre d'interêt en commun
//Dans le cadre du critère "NEED_ONE_HOBBY"
public int countCommonHobbies(){
        ArrayList<String> hostHobbies = this.host.getHobbies
           ();
        ArrayList<String> guestHobbies = this.guest.
           getHobbies();
        int commonHobbies = 0;
        for (String hobby : hostHobbies) {
                if (guestHobbies.contains(hobby)) {
                        commonHobbies++;
                }
        }
        return commonHobbies;
}
```

Cette méthode est utilisée afin de satisfaire ou non le critère `NEED_ONE_HOBBY`.

2. Adaptation de `isCompatible`

Java

```java
public boolean isCompatible() {
        ...
        ...
        String hostHobbies = this.host.
            getCriteriaValue(Criteria.NEED_ONE_HOBBY)
            ;
        String guestHobbies = this.guest.
            getCriteriaValue(Criteria.NEED_ONE_HOBBY)
            ;
        if(!this.host.getHobbies().isEmpty() && !
            this.guest.getHobbies().isEmpty() && (
            hostHobbies.equals("true") ||
            guestHobbies.equals("true")) &&
            countCommonHobbies() < 1){
                return false;
        }
        return true;
}
```

Nous avons implémenté la gestion de la règle spécifiant que les paires avec une `Person` avec pour pays `France`, il faut au moins un hobby en commun. Pour ce faire, nous appelons la fonction précédemment définie `countCommonHobbies`

2.7 Classes de Test

Les classes `ExchangeTest` ainsi que `PersonTest` ont été mises à jour conformément aux ajouts.

De plus, la classe `CriteriaTest` fait son apparition.

Java

```java
public class CriteriaTest {

        private Person alice;

        @BeforeEach
        void setUp() {
                alice = new Person("Alice", "Smith");
                alice.addCriteriaValue(Criteria.
                   HOST_HAS_ANIMAL, "true");
                alice.addCriteriaValue(Criteria.
                   PREFERENCE_GENDER, "Female");
        }

        @Test
        void testCriteriaTypeValid() {
                assertTrue(Criteria.isCriteriaTypeValid(
                   Criteria.HOST_HAS_ANIMAL, "true"));
                assertFalse(Criteria.isCriteriaTypeValid(
                   Criteria.HOST_HAS_ANIMAL, "yes"));
                assertTrue(Criteria.isCriteriaTypeValid(
                   Criteria.PREFERENCE_GENDER, "male"));
                assertFalse(Criteria.isCriteriaTypeValid(
                   Criteria.COUNTRY_OF_ORIGIN, ""));
        }

        @Test
        void testCriteriaValueValid() {
                assertTrue(Criteria.isCriteriaValueValid(
                   Criteria.PREFERENCE_GENDER, "male"));
                assertFalse(Criteria.isCriteriaValueValid(
                   Criteria.PREFERENCE_GENDER, "femme"));
                assertTrue(Criteria.isCriteriaValueValid(
                   Criteria.DATE_OF_BIRTH, "2000-01-01"));
                assertFalse(Criteria.isCriteriaValueValid(
                   Criteria.DATE_OF_BIRTH, "2020-01-01"));
                assertFalse(Criteria.isCriteriaValueValid(
                   Criteria.DATE_OF_BIRTH, "je suis le 8 mai
                    1212 et je suis mal écrit"));
        }
}
```

3 Rendu 3

4 Rendu 4