

République du Cameroun
Paix –Travail – Patrie

Université de Yaoundé I
Sapientia – Collativia – Cognitio

Ecole Nationale Supérieure
Polytechnique de Yaoundé

Département de génie Informatique



Republic of Cameroon
Peace –Work – Fatherland

University of Yaoundé I
Sapientia – Collativia – Cognitio

National Advanced School of
Engineering of Yaounde

Department of Computer Engineering

TUTORIEL DE REALISATION D'UN SYSTÈME MULTIAGENT DE CABINET MEDICAL

NOMS ET PRENOMS DES MEMBRES DU GROUPE 7 :

ALETANOU LOIC	20P082
DIFFO TAHBOH MANUELLA	22P140
DEUTCHOUA MHOH IVAN PARFAIT	21P490
DONFACK KEUBOU AUDREY HILARY	20P189
KEUMOUO TADAHYA DIROIL	20P172
MBOUMELA SOB ELTON LEWIS	20P522
MFANGAM NDASSA YASMINE AMIRAH	20P283
TCHATCHEUNG DJOUM GILLES BORIS	20P221
TENE FOGANG ZACHARIE IGOR	20P540

Sous la coordination de : Pr. BATCHAKUI BERNABE

ANNEE ACADEMIQUE:
2023/2024

TABLE DES MATIERES

1	INTRODUCTION	3
2	CADRE DU PROJET.....	4
2.1	Contexte	4
2.2	Objectifs	4
3	PRÉREQUIS.....	5
3.1	Connaissances	5
3.2	Outils	5
4	INSTALLATION ET CONFIGURATION DE JADE	6
4.1	Installation	6
4.2	Configuration de Jade.....	8
5	TEST DE CRÉATION D'UN AGENT	13
6	CAS PRATIQUE : SMA DE CABINET MEDICAL	21
6.1	Description et scenario	21
6.2	Structuration du projet.....	22
6.3	Implémentation de chaque agent.....	26
6.3.1	Agent Médecin.....	26
6.3.2	Agent Patient.....	35
6.3.3	Agent Secrétaire	42
6.3.4	Main Container	46
6.4	Exécution.....	47
7	CONCLUSION.....	57
8	RÉFÉRENCES BIBLIOGRAPHIQUES.....	58
9	ANNEXE	59

Table des figures

Figure 1 Site officiel de Jade	6
Figure 2 Processus de téléchargement 1	7
Figure 3 Processus de téléchargement 2	7
Figure 4 Fichiers Extraits.....	8
Figure 5 Création d'un projet java sur IDE IntelliJ Idea 1	8
Figure 6 Création d'un projet java sur IDE IntelliJ Idea 2	9
Figure 7 Création d'un projet java sur IDE IntelliJ Idea 3	9
Figure 8 Configuration de Jade 1	10
Figure 9 Configuration de Jade 2	10
Figure 10 Configuration de Jade 3.....	11
Figure 11 Configuration de Jade 4.....	11
Figure 12 Configuration de Jade 5.....	12
Figure 13 Configuration de Jade 6.....	12
Figure 14 Crédit de fichiers.....	13
Figure 15 Classe MainContainer	16
Figure 16 Class FirstAgent	18
Figure 17 Exécution de MainContainer.java 1	19
Figure 18 Crédit du premier agent	20
Figure 19 Crédit des package 1	23
Figure 20 Crédit des package 2	24
Figure 21 Arborescence du SMA	25
Figure 22 Declaration des getters et des setters	45
Figure 23 Constructeurs de la classe Notebook	45
Figure 24 Exécution MainContainer.java	47
Figure 25 Exécution MainContainer.java 2	48
Figure 26 Exécution MainContainer.java 3	48
Figure 27 Exécution MainContainer.java 4	49
Figure 28 Exécution SecretaryContainer	49
Figure 29 Exécution SecretaryContainer 2	50
Figure 30 Exécution DoctorContainer	50
Figure 31 Exécution DoctorContainer 2	51
Figure 32 Exécution PatientContainer	51
Figure 33 Exécution PatientContainer 2	52
Figure 34 Exécution PatientContainer 3	52
Figure 35 Exécution PatientContainer 4	53
Figure 36 Exécution PatientContainer 5	53
Figure 37 Exécution PatientContainer 6	54
Figure 38 Exécution PatientContainer 7	54
Figure 39 Exécution PatientContainer 8	55
Figure 40 Exécution PatientContainer 9	55
Figure 41 Exécution PatientContainer 10	56
Figure 42 Exécution PatientContainer 11	56

1 INTRODUCTION

Les systèmes multi agents sont devenus des outils incontournables dans le domaine de l'intelligence artificielle. Ils permettent de modéliser et de simuler des interactions complexes entre plusieurs agents autonomes, ouvrant ainsi la voie à de nombreuses applications pratiques. L'objectif de ce tutoriel est de permettre à une personne qui n'a aucune connaissance de la notion des systèmes multi agents de le réaliser bout en bout, d'explorer les fondements des systèmes multi agents et de nous guider pas à pas dans la réalisation d'un projet concret. Pour y parvenir, nous commencerons tout d'abord par examiner les prérequis nécessaires, y compris les connaissances de base et les outils requis. Ensuite, nous aborderons l'installation et la configuration de JADE, qui est une plateforme de développement populaire pour les systèmes multi-agents. Une fois l'environnement prêt, nous nous concentrerons sur l'implémentation d'un agent dans un cas général, en explorant les étapes clés et les bonnes pratiques. Enfin, nous présenterons un cas pratique lié à notre système multi agents qui porte sur la gestion d'un cabinet médical, pour illustrer l'application des concepts appris. Suivez le guide, et plongez dans le monde fascinant des systèmes multi-agents.

2 CADRE DU PROJET

2.1 CONTEXTE

Dans ce guide pratique, nous allons explorer la réalisation d'un système multi-agent en Java en utilisant la bibliothèque JADE. Nous plongerons dans le domaine médical en créant un cabinet virtuel avec des agents tels qu'une secrétaire, un médecin et un patient. Ce guide vous accompagnera pas à pas dans la modélisation et la mise en œuvre de ces agents, ainsi que dans la gestion de leurs interactions au sein d'un environnement simulé. Grâce à JADE, vous serez en mesure de développer un système multi-agent fonctionnel qui facilite la gestion des rendez-vous, la coordination des tâches médicales et la communication entre les différents acteurs. Ce guide pratique vous permettra de découvrir les concepts essentiels des systèmes multi-agents tout en explorant leur application concrète dans le contexte médical.

2.2 OBJECTIFS

À la fin de ce tutoriel, l'apprenant doit être capable de :

- Maîtriser l'utilisation de la bibliothèque JADE pour le développement de systèmes multi-agents en Java.
- Créer un premier agent avec Jade
- Mettre en place la communication et la coordination entre les agents, en utilisant les fonctionnalités de messagerie et les protocoles de JADE.
- Simuler des scénarios réalistes d'un environnement, en intégrant les interactions entre les agents et les actions spécifiques à leur rôle.
- Automatiser les tâches de gestion d'un système à l'aide de JADE.
- Et le meilleur pour la fin, créer un système multi agent de bout en bout du début à la fin.

3 PRÉREQUIS

Pour suivre ce tutoriel, certains éléments sont essentiels.

3.1 CONNAISSANCES

- Avoir des bases en programmation orientée objet avec Java.
- Avoir des bases en système expert.
- Maîtriser les concepts d'agent et les systèmes multi-agents.

3.2 OUTILS

- Java SE Development Kit 8 ou une version ultérieure (JDK).
- IDE (Integrated Development Environment) installé. Pour ce tutoriel, nous avons utilisé IntelliJ IDEA.

4 INSTALLATION ET CONFIGURATION DE JADE

Dans le domaine de la conception des systèmes multi-agents, JADE (Java Agent Development Framework) est un Framework open-source développé en langage Java. Il offre un ensemble d'outils, de bibliothèques et de protocoles pour faciliter la création, le développement et le déploiement de systèmes multi-agents.

4.1 INSTALLATION

Nous commencerons par le téléchargement du fichier jar de JADE sur le site <https://jade.tilab.com/download/jade/>.

Une fois que vous aurez cliqué sur ce lien, vous aurez la figure 1 qui présente le site officiel de Jade.

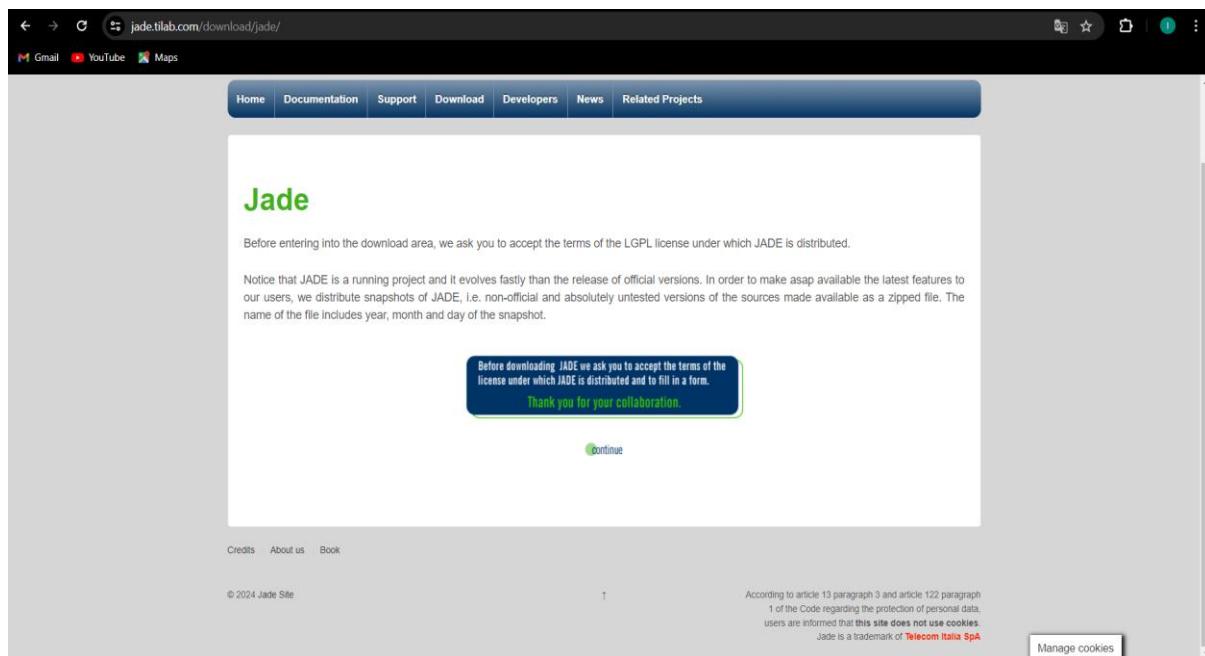


Figure 1 Site officiel de Jade

Pour poursuivre cliquez sur *Continue* et en suivant les instructions comme le montre les figures 2 et 3.



Figure 2 Processus de téléchargement 1

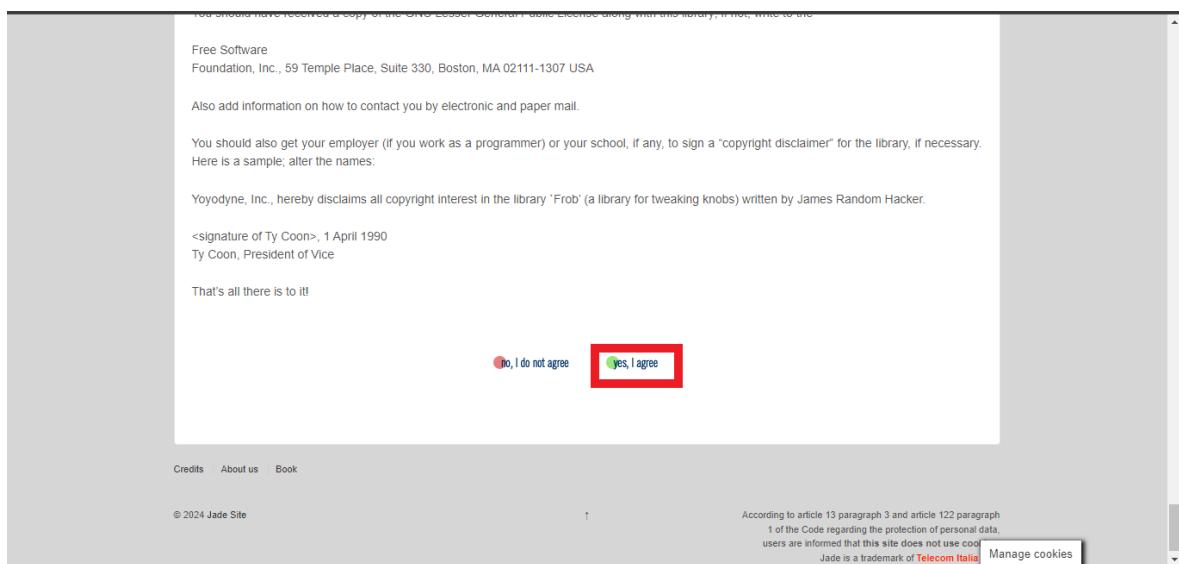


Figure 3 Processus de téléchargement 2

Votre téléchargement débutera après cette action et ne durera que quelques instants. Une fois votre téléchargement terminé, vous allez dans le répertoire ‘Downloads’ de votre machine pour le retrouver.

Extraire le fichier .zip téléchargé dans un dossier

Après avoir extrait le dossier téléchargé, vous aurez les fichiers suivants comme le montre la figure 4.

JADE-bin-4.6.0.zip	02/12/2022 17:36	Archive WinRAR ZIP	2 552 Ko
JADE-doc-4.6.0.zip	02/12/2022 19:01	Archive WinRAR ZIP	13 848 Ko
JADE-examples-4.6.0.zip	02/12/2022 17:35	Archive WinRAR ZIP	505 Ko
JADE-src-4.6.0.zip	02/12/2022 17:35	Archive WinRAR ZIP	2 304 Ko

Figure 4 Fichiers Extraits

Par la suite nous allons extraire le fichier *JADE-bin-4.6.0.zip*.

4.2 CONFIGURATION DE JADE

Ouvrez votre IDE. Ici il s'agit de IntelliJ IDEA.

Créer un nouveau projet Java en cliquant sur l'onglet file, ensuite sur l'onglet new et enfin sur l'onglet Projet comme le montre la figure 5.

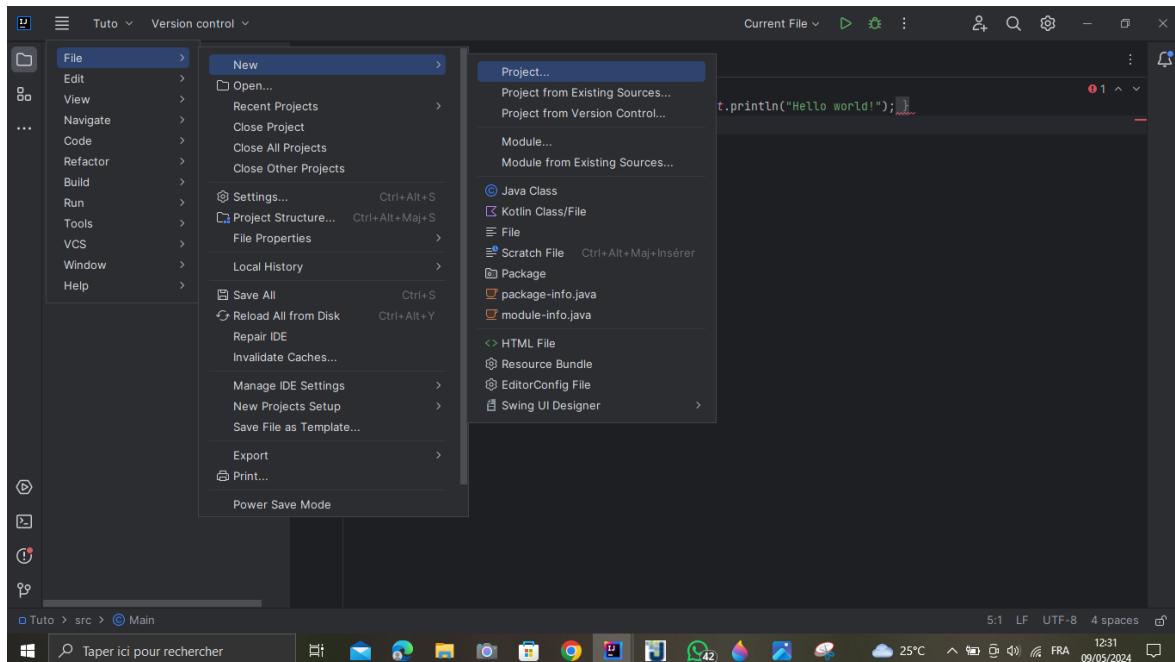


Figure 5 Crédit d'un projet java sur IDE IntelliJ Idea 1

Une boîte de dialogue apparaîtra et vous demandera d'entrer le nom du projet que vous souhaitez créer.

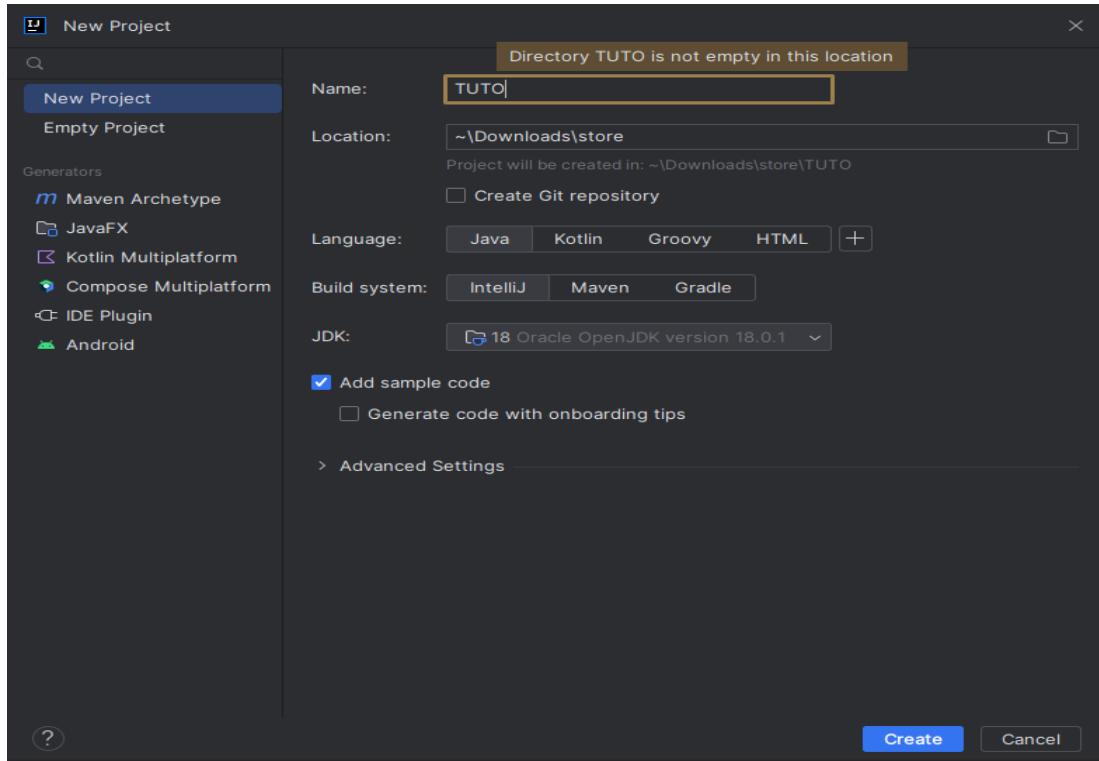


Figure 6 Création d'un projet java sur IDE IntelliJ Idea 2

Cliquer sur *create* comme le montre la figure 7.

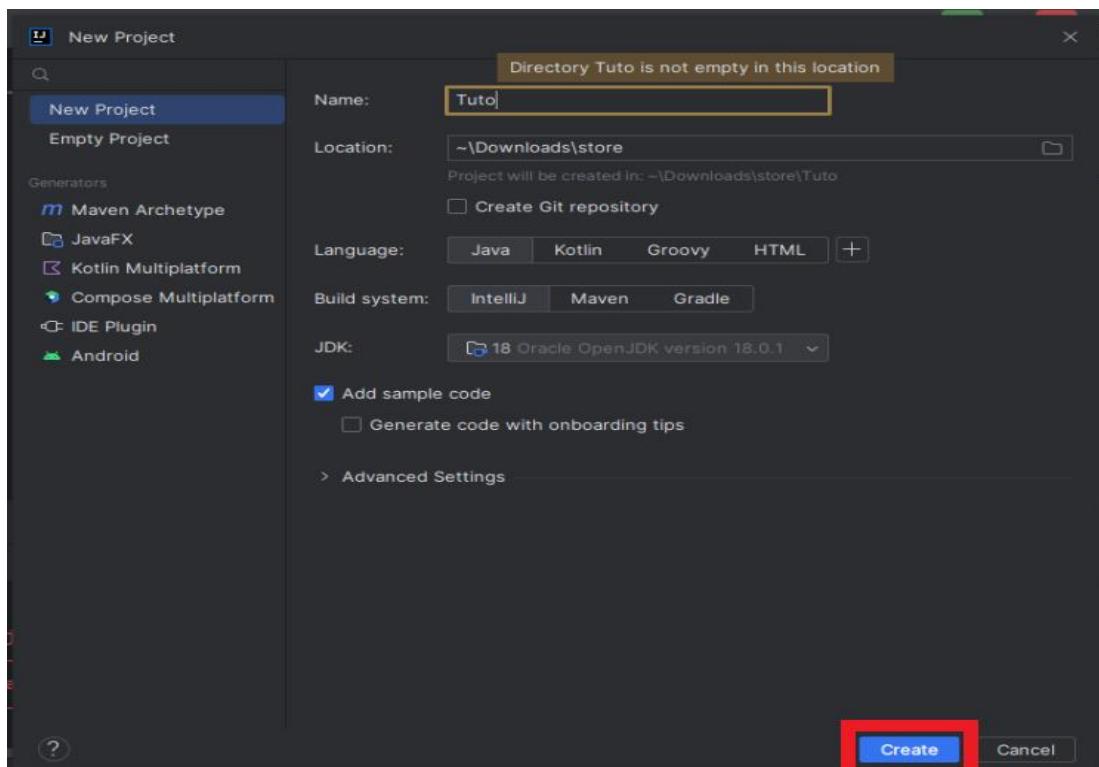


Figure 7 Création d'un projet java sur IDE IntelliJ Idea 3

Ajoutons JADE au projet.

Cliquez sur *File* puis sélectionnez *Project Structure* comme le montre la figure 8.

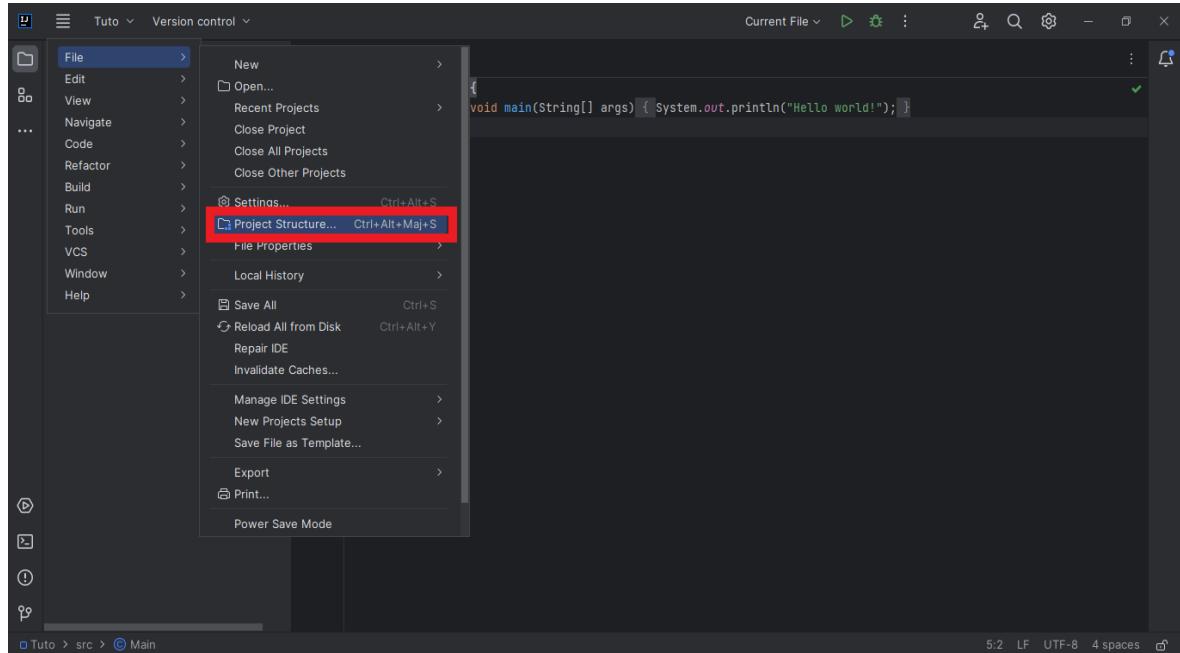


Figure 8 Configuration de Jade 1

Cliquer sur *Libraries* ensuite sur + comme le montre la figure 9

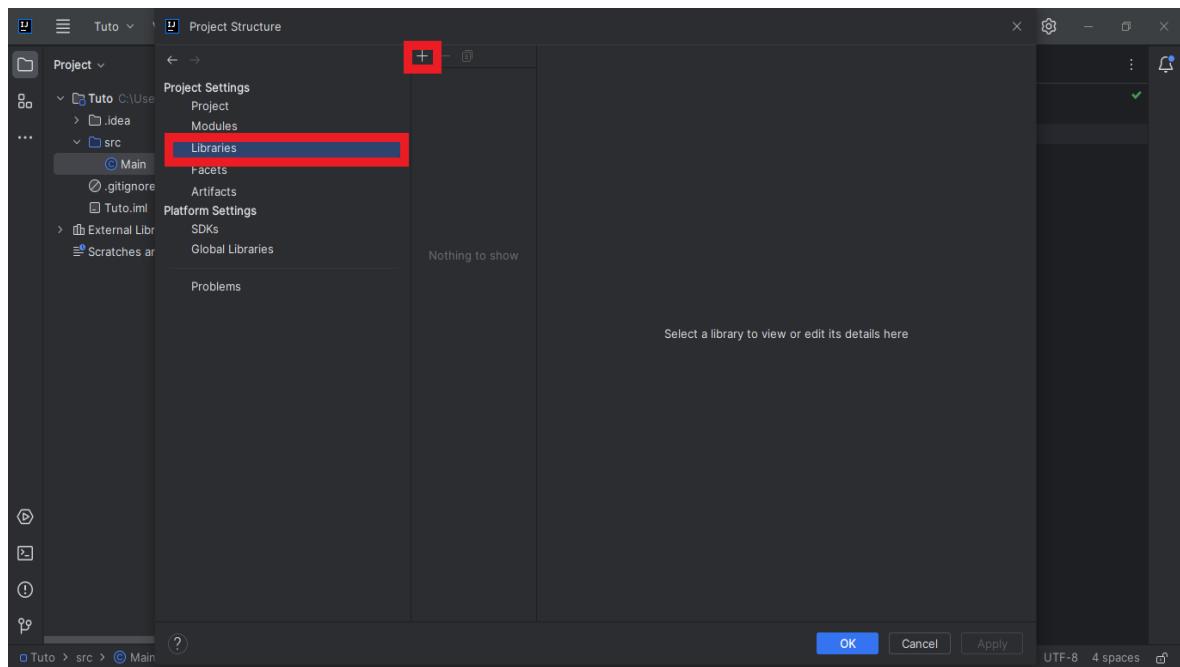


Figure 9 Configuration de Jade 2

Puis sélectionnez le champ *Java* pour ajouter le fichier .jar (*jade.jar*) situé à l'emplacement JADE-bin-4.6.0\jade\lib\jade.jar comme le montre la figure 10.

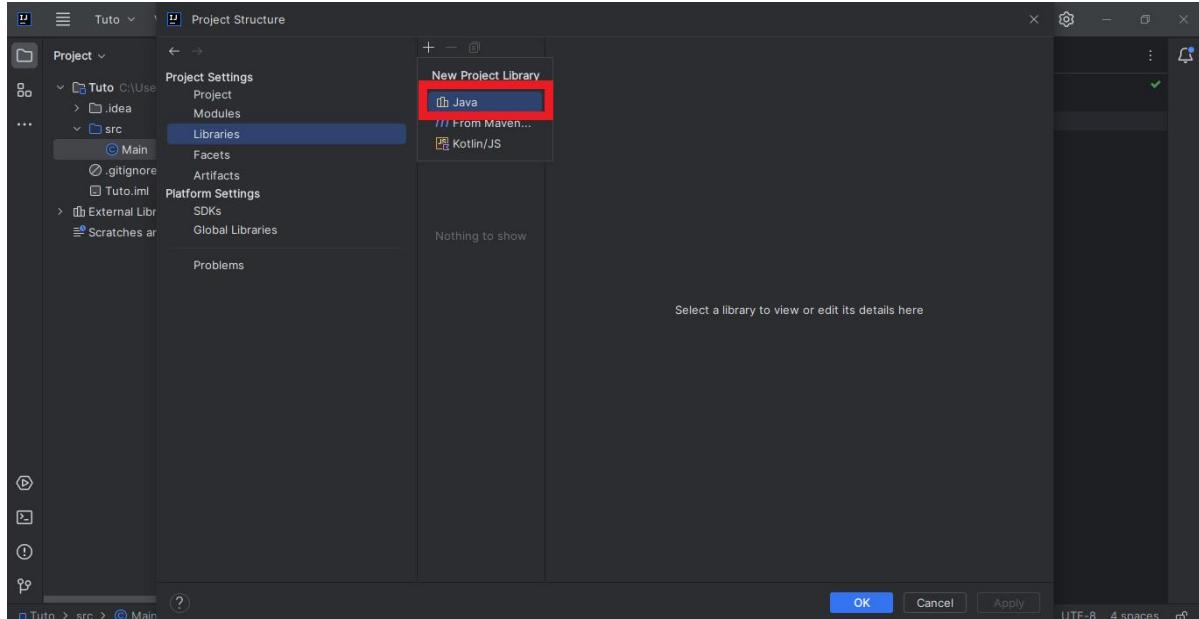


Figure 10 Configuration de Jade 3

Clique sur le bouton *ok* pour ajouter la librairie jade dans ce module comme présenté à la figure 11.

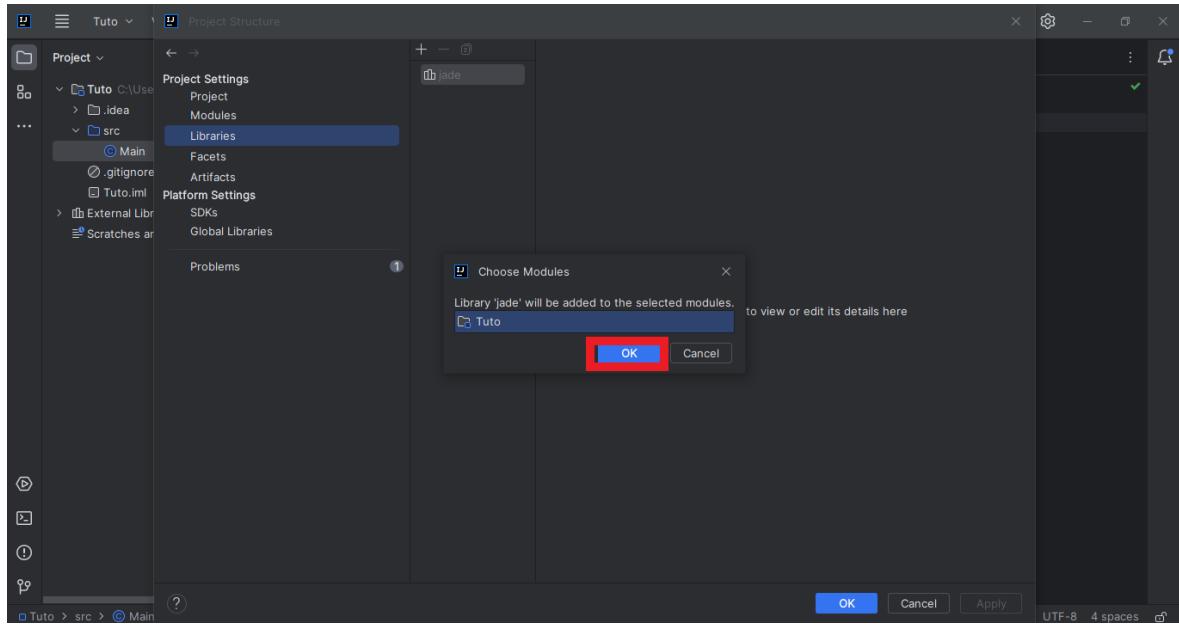


Figure 11 Configuration de Jade 4

Pour terminer et valider ce processus, cliquez sur *Apply* puis sur *OK*.

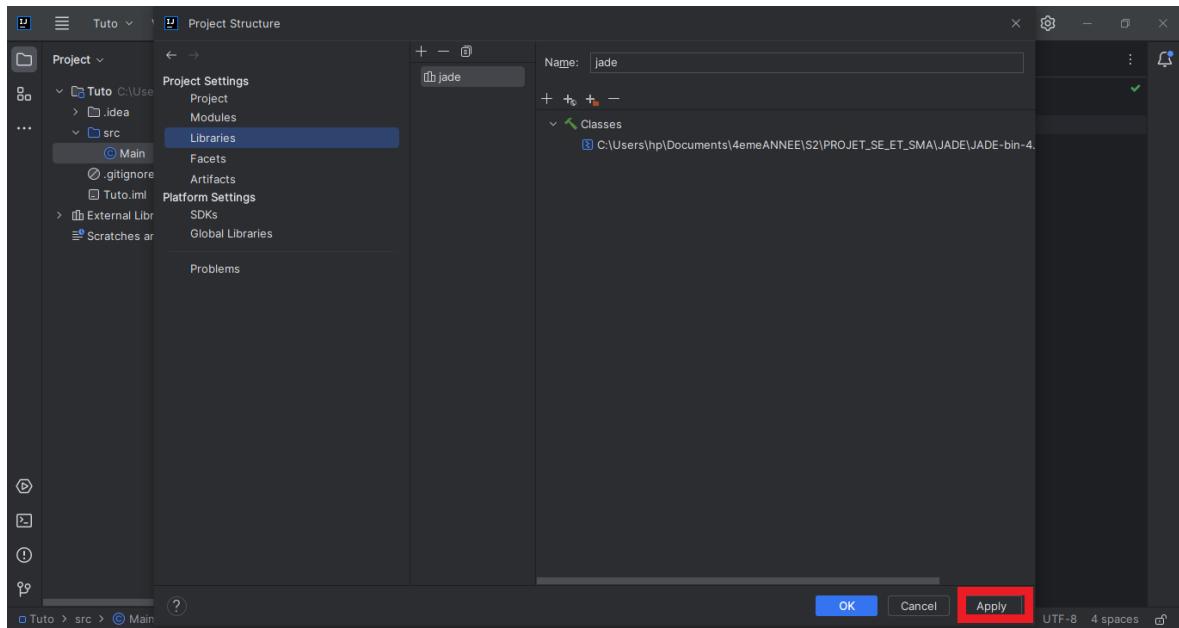


Figure 12 Configuration de Jade 5

En suivant à la lettre ces étapes, Jade sera configuré avec succès dans votre projet et vous verrez dans la rubrique *External Libraries* le dossier jade récemment ajouté en revenant à l'accueil comme présenté à la figure 14.

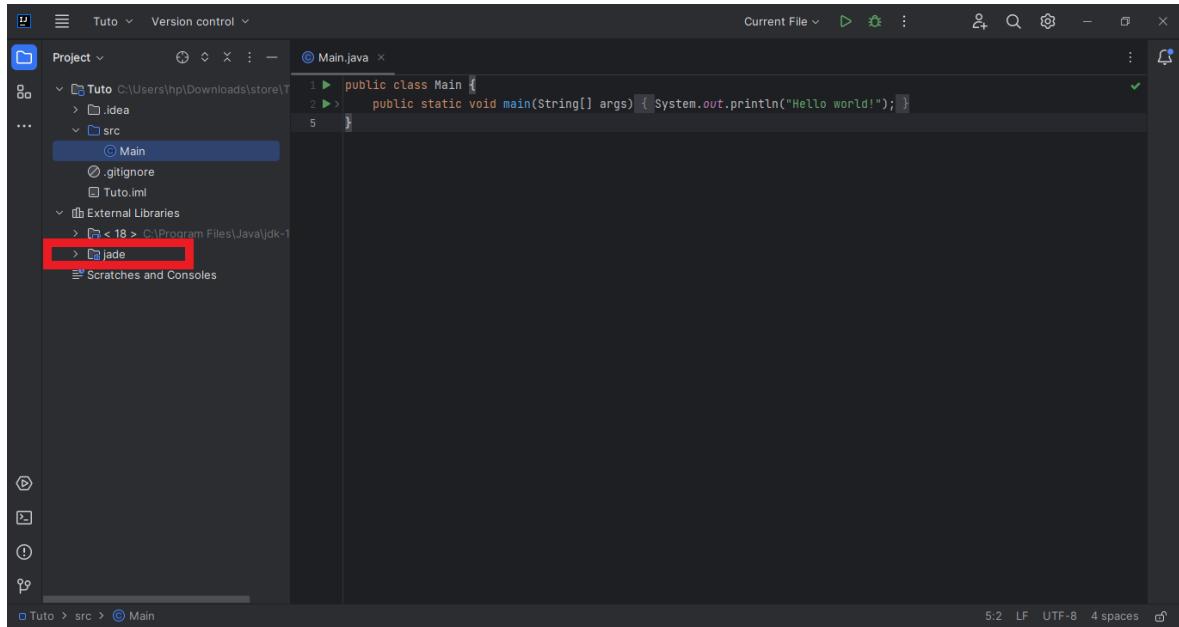


Figure 13 Configuration de Jade 6

5 TEST DE CRÉATION D'UN AGENT

Pour vérifier que la configuration a fonctionné, nous allons créer un agent simple. Pour créer un agent, nous devons créer deux classes : la classe du **conteneur** et la classe de l'**Agent** proprement dit.

Dans le dossier *src* du projet créé dans la première partie, Nous allons créer les fichiers de base qui sont le *MainContainer.java* et *FirstAgent.java*. Mais avant nous supprimerons le fichier Main.java déjà existant pour qu'il n'y ait pas de confusion.

- Créer les fichiers MainContainer.java et FirstAgent.java

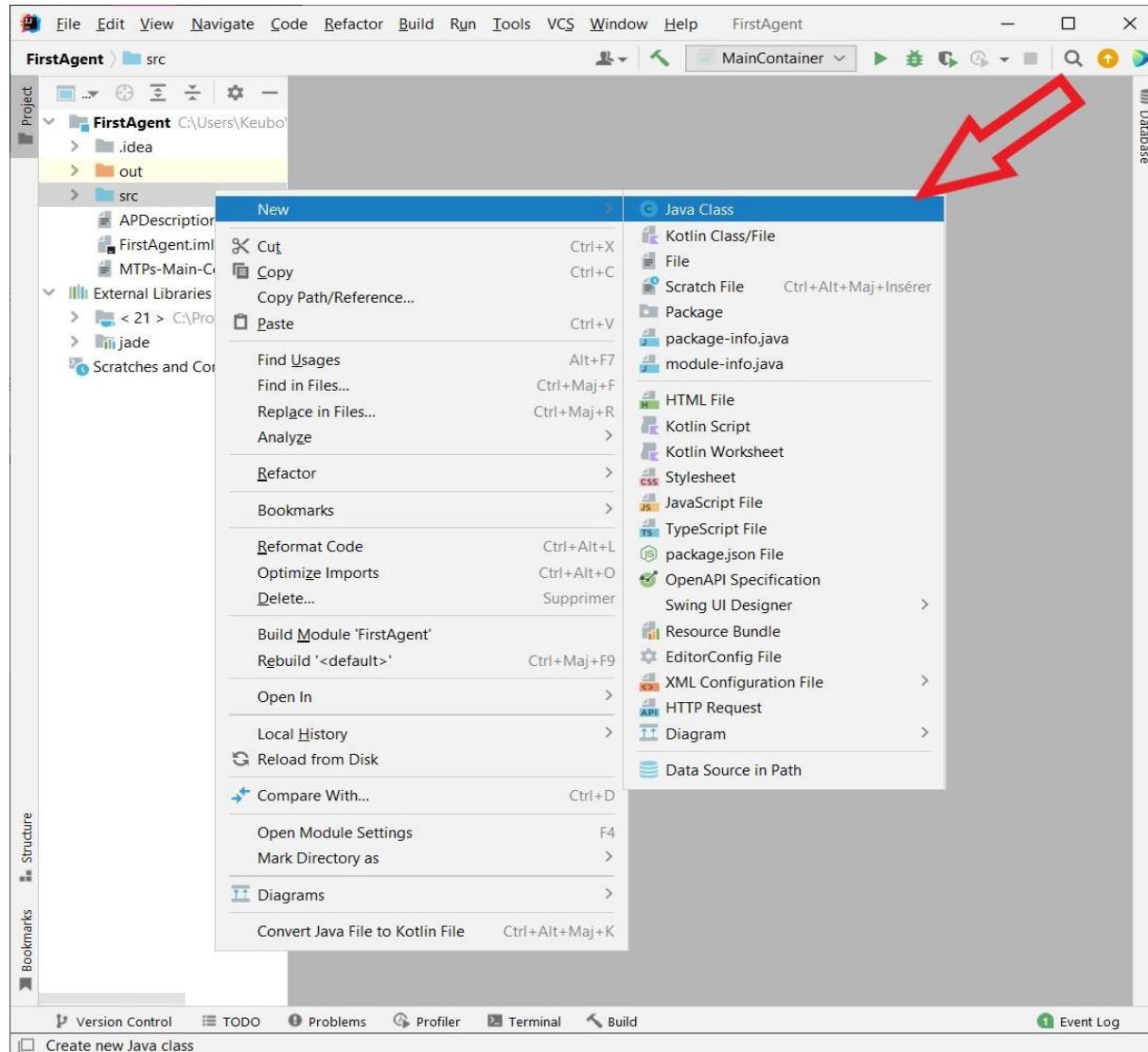


Figure 14 Création de fichiers

- Dans Le fichier **MainContainer.java** , ajouter le code suivant:

```

import jade.wrapper.AgentContainer;
import jade.core.Profile;
import jade.core.Runtime;
import jade.util.ExtendedProperties;
import jade.core.ProfileImpl;
import java.util.Properties;
import jade.wrapper.AgentController;

public class MainContainer {
    public static void main(String[] args) {
        try { Runtime runtime = Runtime.instance();
            Properties properties = new ExtendedProperties();
            properties.setProperty(Profile.GUI, "true");
            Profile profile = new ProfileImpl(jade.util.leap.Properties.toLeapProperties(properties));
            AgentContainer mainContainer = runtime.createMainContainer(profile);

            AgentController agentController = mainContainer.createNewAgent("premier_Agent",
                "FirstAgent.class", new Object[]{});

            mainContainer.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Ce code Java définit une classe nommée MainContainer qui crée et démarre un conteneur pour l'agent FirstAgent.

- **Runtime.getInstance()**: Cette ligne récupère une référence à l'instance unique de l'environnement d'exécution JADE.
- **ExtendedProperties**: Un objet est créé pour contenir les propriétés de configuration.
- **properties.setProperty(Profile.GUI, "true")**: Cette ligne définit une propriété dans l'objet properties pour activer l'interface utilisateur graphique (GUI) de la plateforme JADE
- **ProfileImpl**: Un nouvel objet de profil est créé en utilisant la classe ProfileImpl. Ce profil définit la configuration du conteneur d'agents.
- **jade.util.leap.Properties.toLeapProperties(properties)** : Cette ligne convertit l'objet Java standard Properties en un format compatible avec JADE (leap.Properties).
- **new ProfileImpl(jade.util.leap.Properties.toLeapProperties(properties))** : Cette ligne crée un nouvel objet de profil en utilisant les propriétés converties.
- **runtime.createMainContainer(profile)** : Cette ligne utilise l'environnement d'exécution JADE pour créer un nouveau conteneur principal pour les agents en fonction du profil défini.

Création et démarrage d'un agent :

- **mainContainer.createNewAgent("premier_Agent", "FirstAgent.class", new Object[]{})** : Cette ligne crée un nouvel agent au sein du conteneur.
- **"premier_Agent"** : C'est le nom local de l'agent au sein du conteneur.
- **"FirstAgent.class"** : Ceci spécifie le fichier de classe qui définit le comportement de l'agent (FirstAgent.class). Assurez-vous que ce fichier de classe existe et implémente la classe jade.core.Agent.
- **new Object[]{})** : Il s'agit d'un tableau vide d'arguments passés au constructeur de l'agent.
- **mainContainer.start()** : Cette ligne démarre le conteneur, qui à son tour démarre l'agent créé (premier_Agent).

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help.
- Title Bar:** FirstAgent - MainContainer.java
- Toolbars:** Standard toolbar with icons for Open, Save, Run, Stop, etc.
- Project Tool Window:** Shows the project structure with 'FirstAgent' as the root. Under 'src', there are 'FirstAgent' and 'MainContainer' files, along with 'APDescription.txt', 'FirstAgent.iml', and 'MTPs-Main-Container.txt'. There are also 'External Libraries' and 'Scratches and Consoles' sections.
- Code Editor:** The main window displays the 'MainContainer.java' code. The code imports various Jade classes and defines a main method that creates an AgentContainer, sets its profile properties, and starts it. A warning icon is shown in the gutter at line 10.
- Bottom Status Bar:** Version Control, TODO, Problems, Profiler, Terminal, Build, Event Log, and Build status message: "Build completed successfully in 3 sec, 365 ms (12 minutes ago)".
- Bottom Right:** Encoding: 31:1 CRLF, UTF-8, 4 spaces.

```
import jade.wrapper.AgentContainer;
import jade.core.Profile;
import jade.core.Runtime;
import jade.util.ExtendedProperties;
import jade.core.ProfileImpl;
import jade.wrapper.AgentController;

import java.util.Properties;

public class MainContainer {
    public static void main(String[] args) {
        try { Runtime runtime = Runtime.instance();
            Properties properties = new ExtendedProperties();
            properties.setProperty(Profile.GUI, "true");
            Profile profile = new ProfileImpl(jade.util.leap.Properties.t
            AgentContainer mainContainer = runtime.createMainContainer(pr
            AgentController agentController = mainContainer.createNewAgent(
            mainContainer.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 15 Classe MainContainer

- Dans le fichier *FirstAgent.java*, ajouter le code suivant:

```

import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.domain.introspection.AddedBehaviour;

public class FirstAgent extends Agent {
    protected void setup() {
        System.out.println("Le premier agent a ete cree.");
        // addbehavior permet de creer un comportement
        addBehaviour(new HandleAppointmentRequestBehaviour());
    }
    @Override
    protected void takeDown() {
        System.out.println("Fin de vie");
    }
    private class HandleAppointmentRequestBehaviour extends Behaviour {
        @Override
        public void action() {
        }
        @Override
        public boolean done() {
            return false;
        }
    }
}

```

- La Méthode **setup()** : La fonction *setup()* dans un agent Jade est utilisée pour effectuer les initialisations nécessaires et configurer les comportements de l'agent. Elle est exécutée au moment de la création de l'agent et permet de préparer l'agent à son fonctionnement dans le système multi-agent.
- La Méthode **takedown()**: c'est la méthode exécutée par l'agent en fin de vie. Autrement dit, si on arrête l'agent , l'agent exécutera le contenu de cette fonction avant de se stopper.
- Un **behaviour** (ici, c'est le *handleAppointmentRequestBehaviour*): c'est un comportement que l'agent aura. L'agent peut avoir plusieurs behaviour. On y reviendra. Mais il faut garder à l'esprit que dans ce type de méthode ,il est obligatoire de redéfinir la méthode **action()**. **En effet**, c'est le contenu de cette méthode qui est exécuté lorsque le behaviour est lancé.

```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help FirstAgent - FirstAgent.java
FirstAgent > src > FirstAgent
Project MainContainer Database
FirstAgent C:\Users\Keubo\ FirstAgent.java
src
  FirstAgent
  MainContainer
  APDescription.txt
  FirstAgent.iml
  MTPs-Main-Container.tx
External Libraries
  < 21 > C:\Program Files
  jade
Scratches and Consoles
Database
FirstAgent.java
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.domain.introspection.AddedBehaviour;

public class FirstAgent extends Agent {
    protected void setup() {
        System.out.println("Le premier agent a été créé.");
        // addbehavior permet de créer un comportement
        addBehaviour(new HandleAppointmentRequestBehaviour());
    }

    @Override
    protected void takeDown() {
        System.out.println("Fin de vie");
    }

    private class HandleAppointmentRequestBehaviour extends Behaviour {
        @Override
        public void action() {
        }

        @Override
        public boolean done() {
            return false;
        }
    }
}

```

Figure 16 Class FirstAgent

- Nous pouvons maintenant lancer le test. Pour exécuter un agent, il faut toujours commencer par la classe conteneur de celui-ci. Ici, elle est représentée par MainContainer.

Exécutons le fichier MainContainer.java:

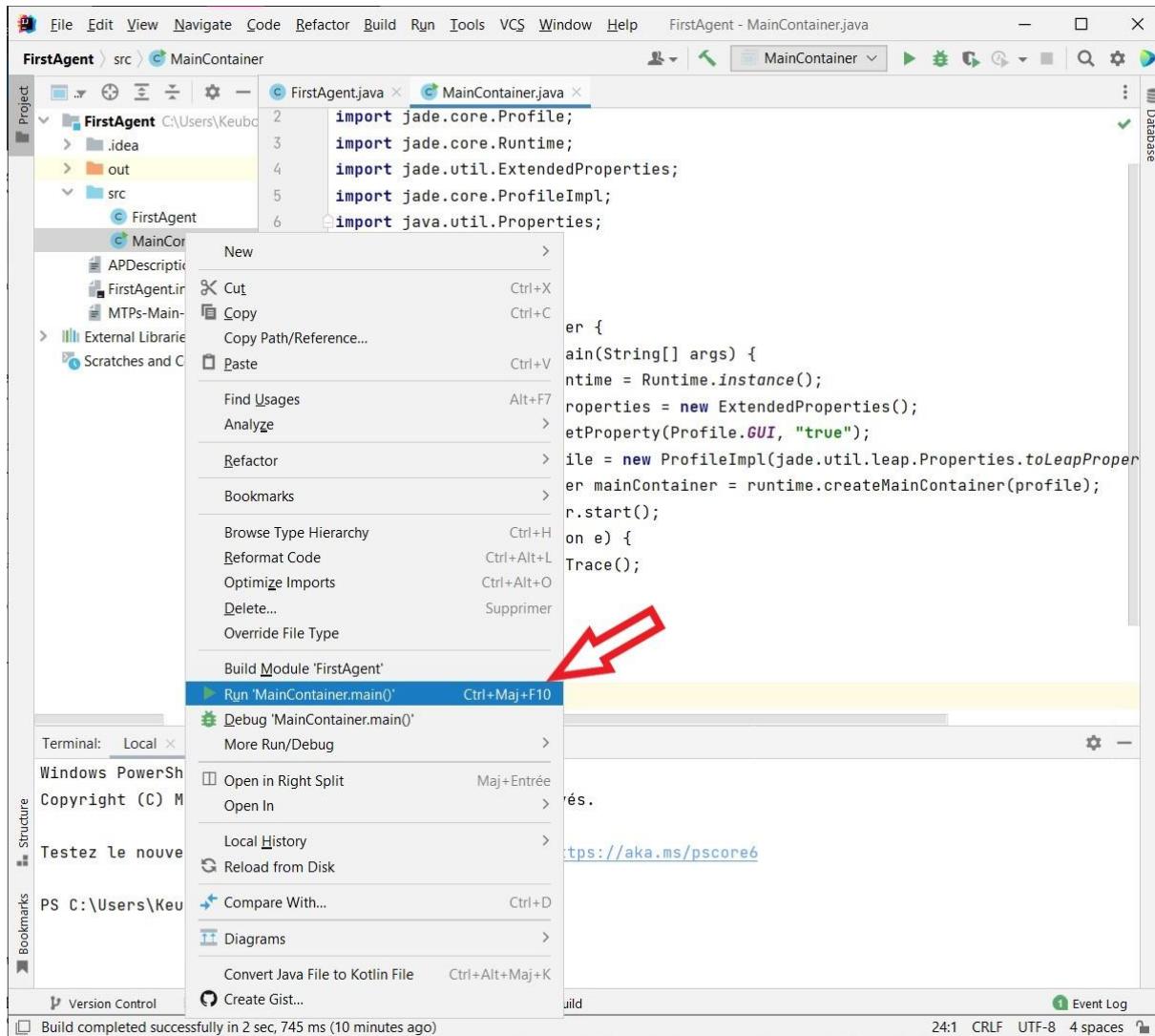


Figure 17 Exécution de MainContainer.java 1

Après exécution, sur le terminal on peut lire “Agent container Main-Container@192.168.56.1 is ready”

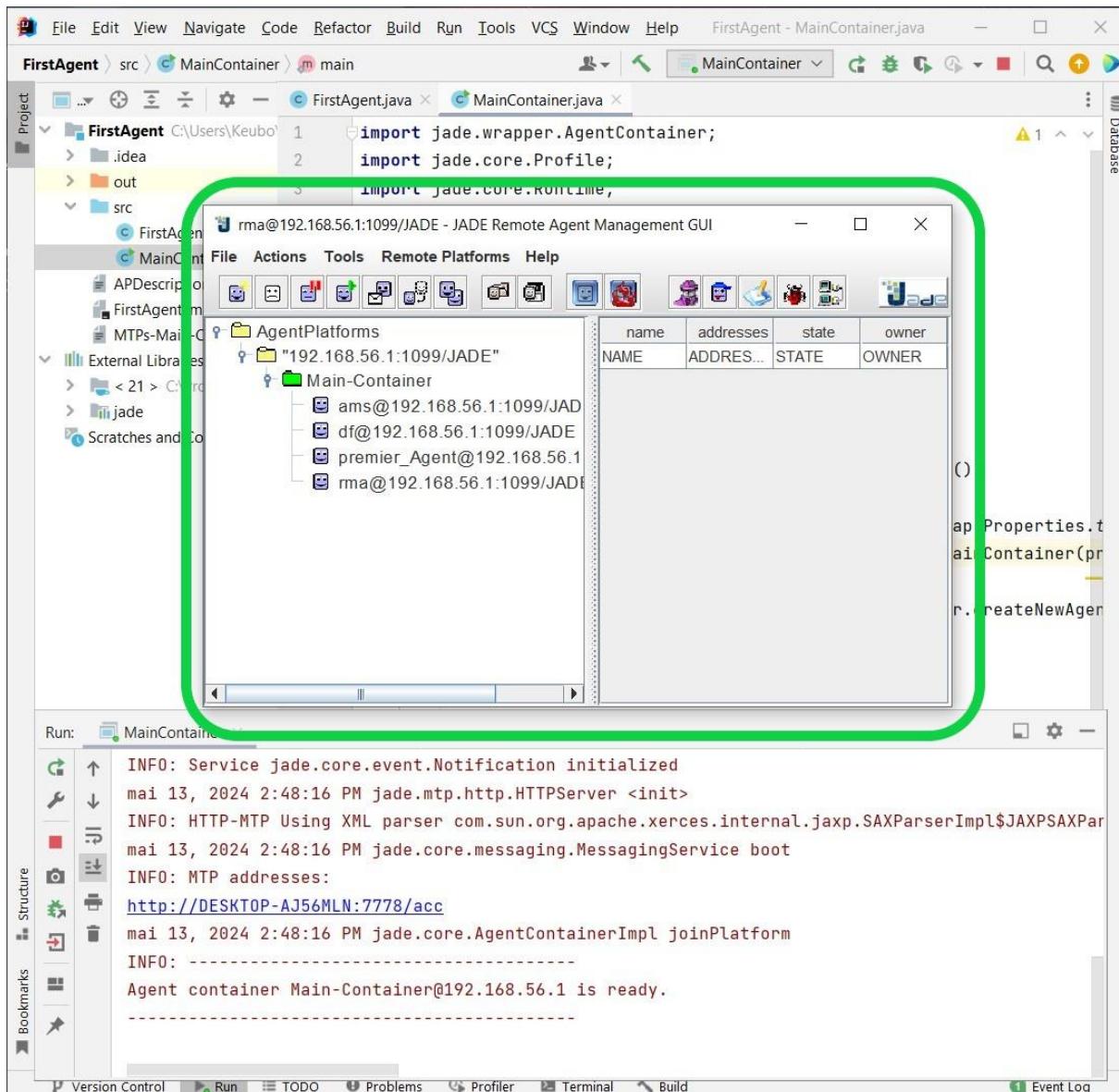


Figure 18 Création du premier agent

Nous avons également l’interface de JADE qui s’est ouverte, indiquant la création de l’agent nommé premier_Agent.

La création de ce premier agent a été un succès. Nous pouvons désormais réaliser un véritable système multi-agent.

6 CAS PRATIQUE : SMA DE CABINET MEDICAL

Dans le cas pratique implémenter un système de gestion d'un cabinet médical.

Le code complet est fourni en annexe.

6.1 DESCRIPTION ET SCENARIO

Le cabinet médical est un lieu où les patients se rendent pour obtenir des soins médicaux. Il est doté d'une salle d'attente où les patients peuvent patienter avant leur rendez-vous avec le médecin.

À l'entrée du cabinet médical, il y a une réception avec une secrétaire. La secrétaire joue un rôle essentiel dans l'organisation des rendez-vous entre les patients et le médecin. Les patients arrivent et demandent un rendez-vous avec le médecin à la secrétaire. Celle-ci consulte le médecin pour connaître sa disponibilité et organise le rendez-vous du patient en fonction des créneaux horaires disponibles.

Si le médecin est disponible à un moment donné, la secrétaire fixe le rendez-vous du patient à ce moment-là. Si le médecin n'est pas disponible, la secrétaire vérifie si le patient est disposé à attendre. Si le patient accepte, il attend dans la salle d'attente jusqu'à ce que le médecin soit disponible. Sinon, si le patient ne souhaite pas attendre, il quitte le cabinet médical.

Une fois que le rendez-vous est convenu, la secrétaire interpelle le patient dans la salle d'attente. Le patient suit les instructions de la secrétaire et se dirige vers le médecin disponible. Le médecin accueille le patient dans son cabinet et commence une série de questions et réponses pour diagnostiquer la condition médicale du patient. Le médecin pose des questions pertinentes et le patient y répond.

En fonction du diagnostic, le médecin peut prescrire une ordonnance au patient. Le patient consulte l'ordonnance qui lui a été prescrite.

Une fois la consultation terminée, le patient remercie le médecin et la secrétaire pour leurs services et quitte le cabinet médical.

Pour sa réalisation nous allons implémenter trois agents à savoir le Médecin(**docteur**), le **patient** et la **secrétaire** qui sont les acteurs principaux de notre cabinet médical.

6.2 STRUCTURATION DU PROJET

Notre implémentation va reposer essentiellement sur quatre packages à savoir :

- **DOCTOR**
 - DoctorAgent
 - DoctorContainer
- **PATIENT**
 - PatientAgent
 - PatientContainer
- **SECRETARY**
 - SecretaryAgent
 - SecretaryContainer
 - Notebook (nous expliquerons plus loin à quoi servira cette classe java)
- **MainContainer** qui est le conteneur principal

Créer les package nécessaires :

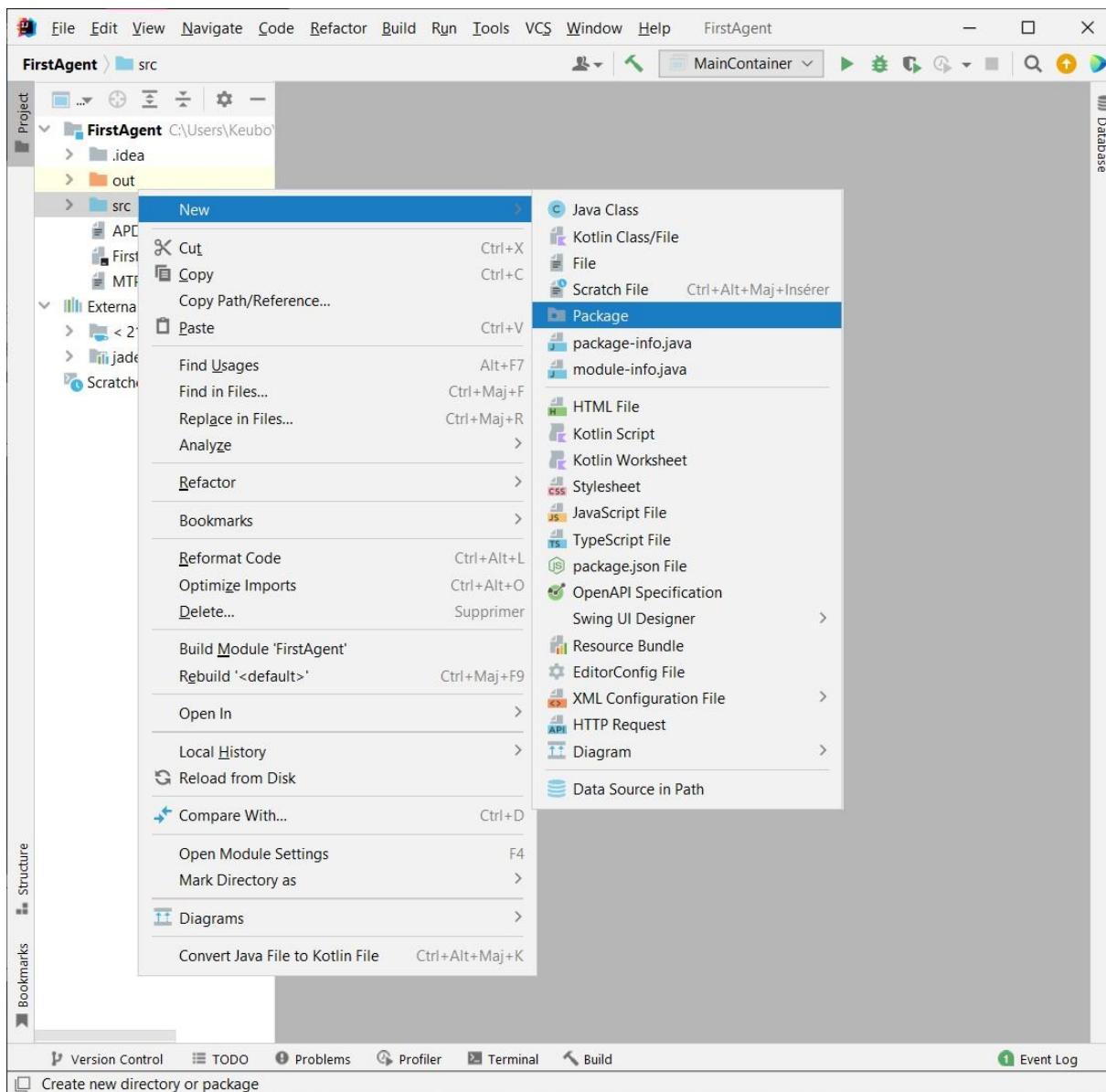


Figure 19 Création des package 1

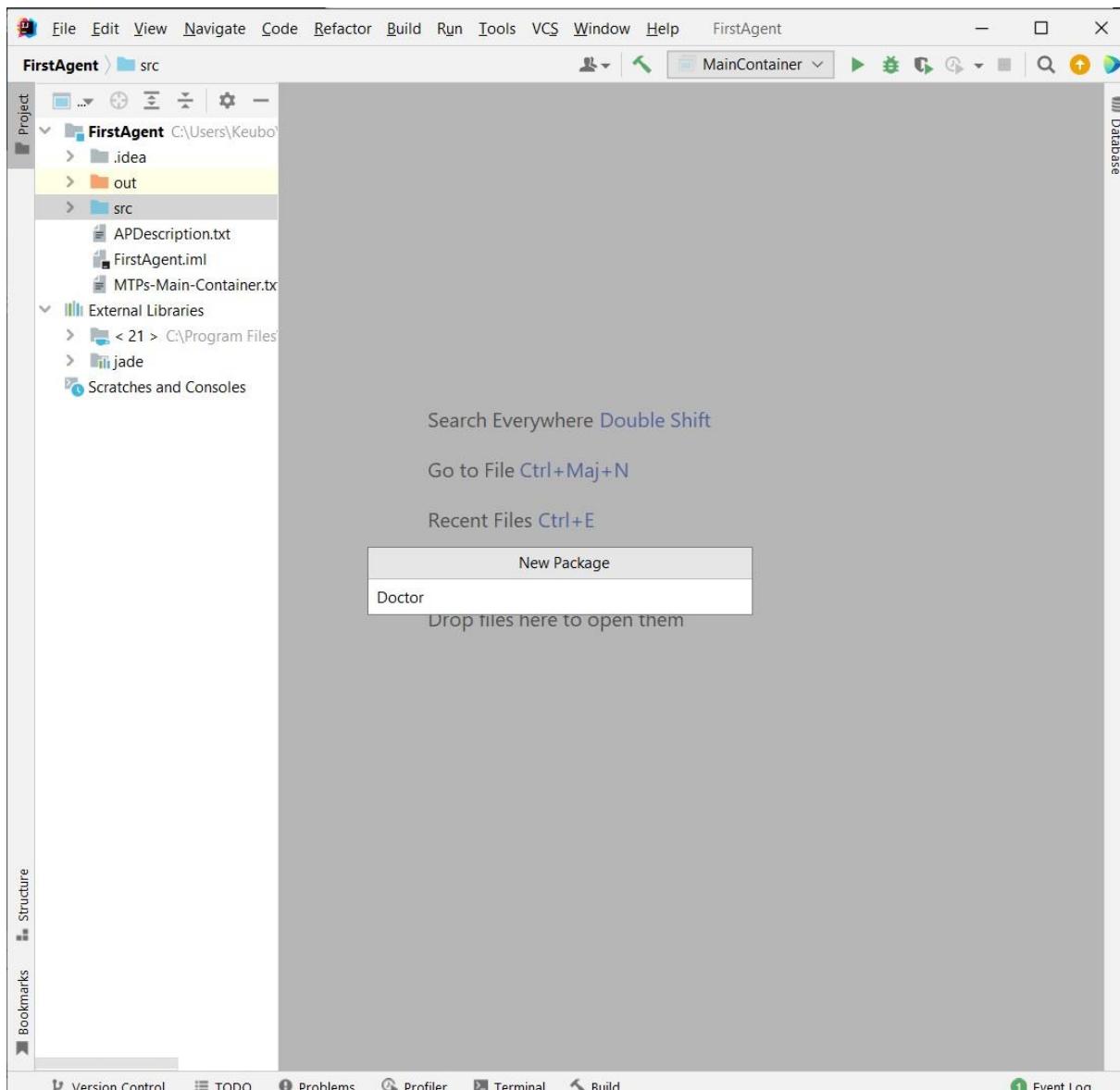


Figure 20 Création des package 2

Nous allons créer les package avec, respectivement les classes suivantes:

- Doctor contenant les classes **Doctor.java** et **DoctorContainer.java**
- Secretary contenant les classes **Secretary.java** et **SecretaryContainer.java**
- Patient contenant les classes **Patient.java** et **PatientContainer.java**

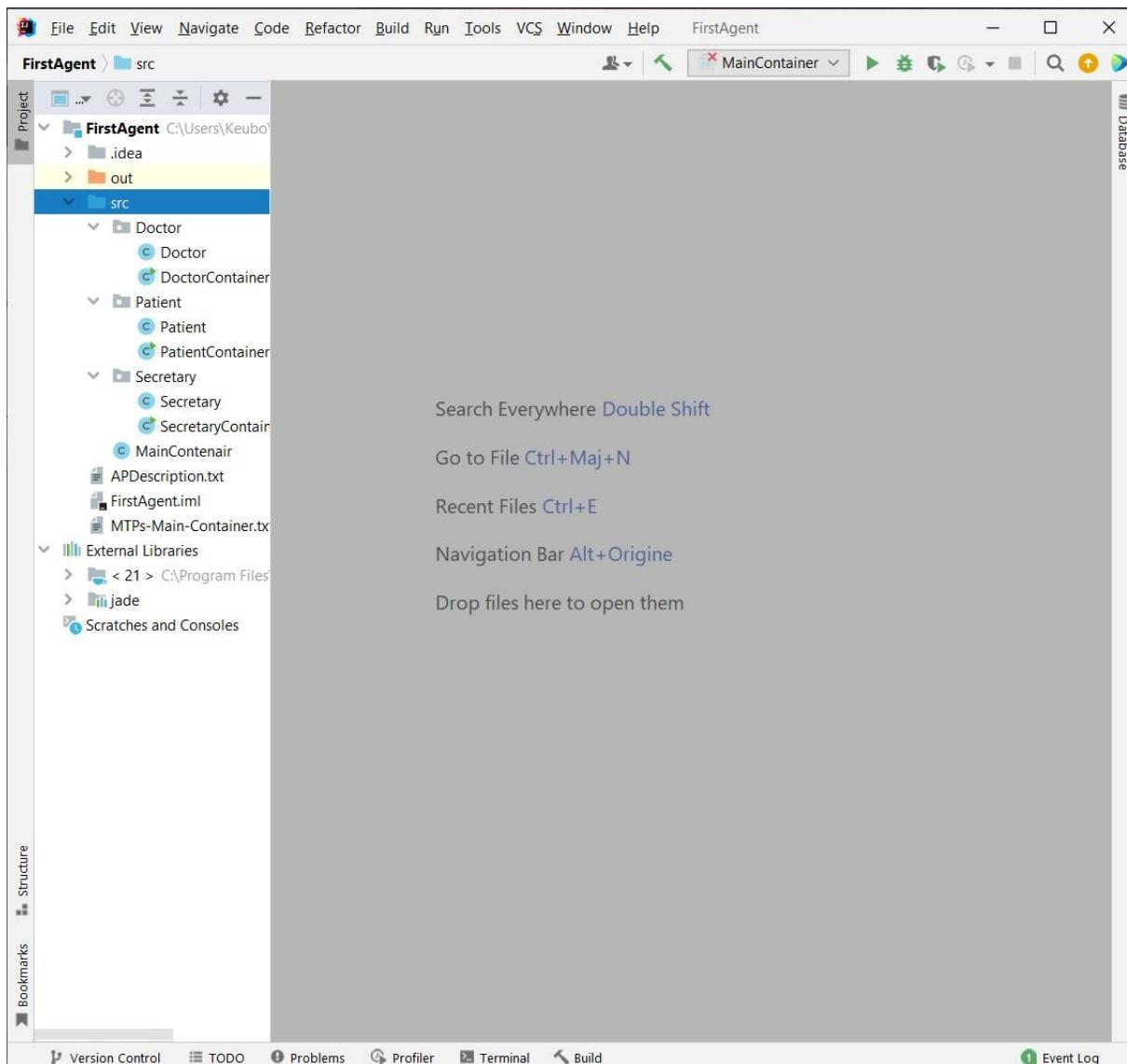


Figure 21 Arborescence du SMA

6.3 IMPLÉMENTATION DE CHAQUE AGENT

6.3.1 Agent Médecin

6.3.1.1 *DoctorContainer*

Pour exécuter l'agent "Doctor", nous devons créer un conteneur Jade et démarrer l'agent à l'intérieur du conteneur. Le contenu de la classe DoctorContainer fourni ci-dessous :

```
public class DoctorContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Profile profile = new ProfileImpl( isMain: false);
        profile.setParameter(Profile.MAIN_HOST, s1: "localhost");
        AgentContainer agentContainer = runtime.createAgentContainer(profile);
        AgentController agentController = agentContainer.createNewAgent( nickname: "medecin1", className: "Doctor", new Object[]{});
        agentController.start();
    }
    no usages
    private static void startAgent(AgentContainer container, String agentName, Class agentClass) throws StaleProxyException {
        AgentController agentController = container.createNewAgent(agentName, agentClass.getName(), args: null);
        agentController.start();
    }
}
```

EXPLICATION

Dans ce code, nous utilisons la classe DoctorContainer pour créer un conteneur Jade et démarrer l'agent "Doctor" à l'intérieur du conteneur. Voici une explication des étapes :

- **Importation des classes nécessaires** : Nous importons les classes nécessaires de JADE pour créer le conteneur et l'agent.
- **Méthode startAgent()** : Cette méthode est utilisée pour démarrer un agent à l'intérieur du conteneur. Elle prend en paramètres le conteneur, le nom de l'agent, la classe de l'agent et les arguments à passer à l'agent.
- **Méthode main()** : Cette méthode est le point d'entrée de notre programme. Elle initialise l'environnement JADE en créant une instance de Runtime et en définissant un Profile avec le paramètre *MAIN_HOST* pour spécifier l'hôte principal (dans cet exemple, "localhost"). Ensuite, nous créons un conteneur d'agents en utilisant le Profile et la méthode *createAgentContainer()*. Enfin, nous démarrons l'agent "Doctor" en appelant la méthode *startAgent()* avec les paramètres appropriés.

6.3.1.2 DoctorAgent

Dans le fichier DoctorAgent,

- **Importation des bibliothèques nécessaires :** Tout d'abord, vous devez importer les bibliothèques nécessaires de JADE et Java pour pouvoir utiliser les fonctionnalités d'agents et de messages.

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.Scanner;
```

- **Définition des symptômes pour chaque maladie :** Dans cette partie, nous définissons des listes de symptômes pour différentes maladies telles que la grippe, le rhume, les allergies, le paludisme, la fièvre typhoïde et le choléra. Ces listes sont utilisées plus tard pour diagnostiquer les patients.

En effet, en l'absence d'un agent laboratoire pour la réalisation d'examens afin de permettre au médecin d'effectuer un diagnostic, ces structures servent à notre agent de base de connaissances. Le but c'est de proposer au patient une liste de symptômes parmi lesquels, il choisit les symptômes qu'il ressent. En fonction de l'appartenance de ces symptômes à une de nos structures ou pas, l'agent médecin répondra automatiquement.

```
---g-
private static final List<String> coldSymptoms = Arrays.asList("nez qui coule", "éternuements", "mal de gorge", "fatigue", "maux de tête");
1 usage
private static final List<String> malariaSymptoms = Arrays.asList("fièvre", "frissons", "maux de tête", "fatigue", "nausées", " vomissements");
1 usage
private static final List<String> typhoidSymptoms = Arrays.asList("fièvre élevée", "maux de tête", "fatigue", "perte d'appétit", "diarrhée");
1 usage
```

- **Déclaration des variables et création de l'agent :** Vous déclarez les variables nécessaires, telles que le scanner pour la saisie utilisateur, les AID pour la secrétaire et le patient, et un booléen pour indiquer si le patient est présent. La déclaration se fait généralement comme à la figure 24.

```
private AID secretary=new AID("infirmière", AID.ISLOCALNAME);
private AID patient;
Scanner sc= new Scanner(System.in);
boolean isPatient=false;
```

- **Définition des méthodes utilitaires :** Ici nous avons le recensement de toutes les méthodes que devra implémenter l'agent DOCTOR. Tout cela est consigné dans le tableau ci-après.

MÉTHODE	DESCRIPTION
informAvailability	Pour envoyer une réponse de départ à un patient(une salutation) et informer la disponibilité du médecin à la secrétaire;A l'aide d'objet ACLMessage de type ACLMessage.INFORM.
extractInformation	L'agent <i>Secretary</i> envoie le carnet médical du patient au médecin. La méthode va extraire des informations spécifiques du contenu du message reçu.
containsAllSymptoms	Cette méthode va vérifier si une liste de symptômes contient tous les symptômes requis pour une maladie particulière.
performDiagnosis	Elle effectue un diagnostic en fonction des symptômes. La méthode prend en entrée une liste de symptômes (ceux entrés par le patient). Elle vérifie la présence de symptômes spécifiques dans les listes de symptômes prédéfinies et renvoie un message de diagnostic.

sendGoodbyeResponse	Cette méthode crée un message d'informations de type "goodbye-response", l'envoie au patient. Cela permettra à l'agent médecin de communiquer avec l'agent patient qu'il quitte la conversation ou la consultation médicale. A l'aide d'objet ACLMessage de type ACLMessage.INFORM.
----------------------------	---

L'implémentation de ces méthodes est détaillée comme suit:

Nous commencerons par les méthodes les plus faciles à comprendre.

1. sendGoodbyeResponse

```
private void sendGoodbyeResponse(AID patient) {
    ACLMessage response = new ACLMessage(ACLMessage.INFORM);
    response.setConversationId("goodbye-response");
    response.addReceiver(patient);
    response.setContent("Doctor");
    send(response);
}
```

Cette méthode sert à envoyer un message du type “au revoir” au patient lorsque la consultation est terminée.

Commentaire :

- **ACLMensaje response = new ACLMessage(ACLMessage.INFORM);**

Cette ligne crée une nouvelle instance de la classe ACLMessage en utilisant le constructeur qui prend un paramètre de type int. Dans cet exemple, le paramètre ACLMessage.INFORM (une performative, pour plus d'information, consulter la documentation) est utilisé pour spécifier le type de message comme une information à envoyer.

- **response.setConversationId("goodbye-response") :** Cette ligne définit l'identifiant de conversation du message en utilisant la méthode setConversationId(). L'identifiant de conversation permet de regrouper plusieurs messages liés à une même conversation.

- **response.addReceiver(patient)** : Cette ligne ajoute un destinataire au message en utilisant la méthode addReceiver(). Dans cet exemple, le destinataire est une variable nommée patient, qui représente l'agent patient auquel le message sera envoyé.
- **response.setContent("Doctor")** : Cette ligne définit le contenu du message en utilisant la méthode setContent(). Dans cet exemple, le contenu du message est simplement défini comme une chaîne de caractères "Doctor".
- **send(response)** : Cette ligne envoie le message en utilisant la méthode send(). Le message est envoyé au(x) destinataire(s) spécifié(s) précédemment.

2. containsAllSymptoms

```
private boolean containsAllSymptoms(List<String> symptoms, List<String> requiredSymptoms) {
    for (String symptom : requiredSymptoms) {
        if (!symptoms.contains(symptom)) {
            return false;
        }
    }
    return true;
}
```

3. performDiagnosis

```
private String performDiagnosis(List<String> symptoms) {  
    if (containsAllSymptoms(symptoms, fluSymptoms)) {  
        return "Vous pourriez avoir la grippe.";  
    } else if (containsAllSymptoms(symptoms, coldSymptoms)) {  
        return "Vous pourriez avoir un rhume.";  
    } else if (containsAllSymptoms(symptoms, allergiesSymptoms)) {  
        return "Vous pourriez avoir des allergies.";  
    } else if (containsAllSymptoms(symptoms, malariaSymptoms)) {  
        return "Vous pourriez avoir le paludisme.";  
    }else if (containsAllSymptoms(symptoms, typhoidSymptoms)) {  
        return "Vous pourriez avoir la fièvre typhoïde."}  
    else if (containsAllSymptoms(symptoms, choleraSymptoms)) {  
        return "Vous pourriez avoir le choléra.";  
    }  
    else {  
        return "Nous ne pouvons pas conclure à un diagnostic dans l' immédiat. " +  
            "svp vous devez vous rendre à l'hôpital pour une analyse approfondie";  
    } }
```

4. InformAvailability

```
private void informAvailability(boolean bool) {  
  
    ACLMessage response = new ACLMessage(ACLMessage.INFORM);  
    response.setConversationId("DOCTOR");  
    response.addReceiver(secretary);  
    String input;  
    if(bool){  
        System.out.println("Secrétaire: Dr svp Etes-vous libre? 1oui, 2.non");  
        input= sc.nextLine();  
    }  
    else{  
        input="1";  
    }  
  
    if(Objects.equals(input, "1")){  
  
        response.setContent("Doctor is available");  
    }  
    else{  
        response.setContent("Doctor is not available");  
    }  
    send(response);  
}
```

5. extractInformation

```
private String extractInformation(String content, String key){  
    String champRecherche = "\"" +key+ ":";  
    int indexDebut = content.indexOf(champRecherche);  
    if (indexDebut == -1) {  
        return null; // Champ non trouvé  
    }  
  
    indexDebut += champRecherche.length();  
    int indexFin = content.indexOf("}", indexDebut);  
  
    if (indexFin == -1) {  
        indexFin = content.indexOf("}", indexDebut);  
    }  
  
    if (indexFin == -1) {  
        return null; // Fin de champ non trouvé  
    }  
  
    return content.substring(indexDebut, indexFin).replace("\"", "").trim();  
}
```

- La méthode **setup ()**

C'est dans cette méthode que nous définissons le comportement de base de l'agent médecin. Ici, le comportement à l'initialisation de l'agent est donné par le behaviour **HandleSecretaryMessageBehaviour** comme illustré ci-dessous.

```
protected void setup() {
    addBehaviour(new HandleSecretaryMessagesBehaviour());
}

protected void takeDown() {
    // Perform cleanup tasks here
}
```

- **Classe interne HandleSecretaryMessagesBehaviour** : Cette classe hérite de CyclicBehaviour (afin d'exécuter le comportement en permanence) et définit le comportement de l'agent pour traiter les messages reçus de la secrétaire. Dans la méthode action(), nous utiliserons une instruction switch pour traiter différents types de messages en fonction de leur identifiant de conversation. Le code complet de cette méthode est donné en annexe.

```
private class HandleSecretaryMessagesBehaviour extends CyclicBehaviour {
    public void action() {
        ACLMessage msg = receive();
        if (msg != null) {
            switch (msg.getConversationId()) {

                case "askAvailability": ...
                case "patient-arrival": ...
                case "goodbye-from-patient": ...
                case "patient": ...
                case "symptoms": ...
            }
        }
    }
}
```

6.3.2 Agent Patient

6.3.2.1 PatientContainer

Pour exécuter l'agent "Patient", nous devons créer un conteneur Jade et démarrer l'agent à l'intérieur du conteneur. Le contenu de la classe PatientContainer illustre comment réaliser cela. Il suffit de reprendre le code du conteneur (*cf Figure 21*) de l'agent médecin et de modifier juste les paramètres correspondants l'agentName et l'agentClass en mettant respectivement pour le cas d'espèce ici, "**malade**" et "**Patient**".

6.3.2.2 PatientAgent

Cet agent représente un patient qui interagit avec une infirmière et un médecin.

- Importations :

Les importations nécessaires pour le code de l'agent sont effectuées, y compris les classes de Jade.

- Définition de la classe :

La classe PatientAgent est définie en tant qu'agent Jade. Elle hérite de la classe Agent de Jade et remplace certaines de ses méthodes pour définir le comportement de l'agent.

```
public class Patient extends Agent {  
    List<String> symptoms;  
    AID doctor;  
    @Override  
    protected void setup() {  
        System.out.println("Patient Agent " + getAID().getName() + " est prêt");  
  
        // Step 1: Send an appointment request to the secretary  
        ACLMessage request = new ACLMessage(ACLMessage.REQUEST);  
        request.setConversationId("appointment-from-patient");  
        request.setContent("Appointment request");  
        request.addReceiver(new AID("infirmière", AID.ISLOCALNAME));  
        send(request);  
        System.out.println("Le Patient a envoyé une demande de rendez-vous chez l'infirmière");  
  
        // Add behavior for requesting an appointment  
        addBehaviour(new RequestAppointmentBehaviour());  
    }  
}
```

Un comportement (RequestAppointmentBehaviour) est ajouté à l'agent pour gérer la demande de rendez-vous

- **Définition des méthodes utilitaires :** Dans ce tableau nous avons le récapitulatif de toutes les méthodes que devra implémenter l'agent PATIENT

MÉTHODE	DESCRIPTION
Notebook ()	<p>Cette méthode est utilisée pour collecter les informations du patient en utilisant une interaction avec l'utilisateur via la console.</p> <p>Elle demande la taille, le sexe et le poids du patient, puis construit une chaîne JSON contenant les informations du patient.</p> <p>La chaîne JSON est renvoyée en tant que contenu de la méthode</p>
getSymptoms ()	<p>Cette méthode permet au patient de sélectionner les symptômes qu'il ressent en utilisant une interaction avec l'utilisateur via la console.</p> <p>Elle affiche une liste de symptômes et demande à l'utilisateur d'entrer les numéros correspondants aux symptômes qu'il ressent.</p> <p>Les symptômes sélectionnés sont renvoyés sous forme de chaîne séparée par des virgules</p>

1. Notebook()

```
private String Notebook(){

    Scanner scanner = new Scanner(System.in);

    // Variables pour les informations du patient
    System.out.println("Veuillez entrer votre taille :");
    String size = scanner.nextLine();

    System.out.println("Veuillez entrer votre sexe :");
    String gender = scanner.nextLine();

    System.out.println("Veuillez entrer votre poids en kg:");
    String weight = scanner.nextLine();
    // Construire le contenu du message en utilisant la concaténation de chaînes
    String content = "{"
        + "\"size\": \"" + size + "\","
        + "\"gender\": \"" + gender + "\","
        + "\"weight\": \"" + weight + "\","
        + "}";

    return content;
}
```

Vous pouvez si vous le voulez ajouter des éléments comme le groupe sanguin, le nom du patient, son statut, l'assurance, prélever sa température etc. Il suffira d'adapter la chaîne content.

```
// Construire le contenu du message en utilisant la concaténation de chaînes
String content = "{"
    + "\"size\": \"" + size + "\","
    + "\"gender\": \"" + gender + "\","
    + "\"weight\": \"" + weight + "\","
    + "\"bloodType\": \"" + bloodType + "\","
    + "\"patientName\": \"" + patientName + "\","
    + "\"medicalHistory\": \"" + medicalHistory + "\","
    + "\"insurance\": \"" + insurance + "\","
    + "\"contactInfo\": \"" + contactInfo + "\","
    + "\"temperature\": \"" + temperature + "\","
    + "\"bloodPressure\": \"" + bloodPressure + "\","
    + "\"reason\": \"" + reason + "\""
    + "};
```

2. getSymptom()

```
private String getSymptoms(){

    List<String> selectedSymptoms = new ArrayList<>();
    Scanner scanner = new Scanner(System.in);
    while (true) {
        // Affichage du menu
        int counter = 1;
        for (String symptom : symptoms) {
            System.out.printf("%-2d. %-25s", counter, symptom);
            if (counter % 4 == 0) {
                System.out.println();
            }
            counter++;
        }
        System.out.println();
        // Lecture des choix du patient
        String input = scanner.nextLine();
        // Analyse des choix du patient
        String[] choices = input.split(",");
        for (String choice : choices) {
            try {
                int index = Integer.parseInt(choice.trim());
                if ((index >= 1 && index <= symptoms.size()) || index>=30) {
                    String symptom = symptoms.get(index - 1);
                    if (!selectedSymptoms.contains(symptom)) {
                        selectedSymptoms.add(symptom);
                    }
                } else {
                    System.out.println("Numéro de symptôme invalide : " + index);
                }
            }
        }
    }
}
```

```

} catch (NumberFormatException e) {
    System.out.println("Format de numéro invalide : " + choice);
}
System.out.println("Avez-vous terminé? 1. oui ,2. non");
String entry = scanner.nextLine();
if (entry.equals("1")) {
    System.out.println("Symptômes sélectionnés :");
    for (String symptom : selectedSymptoms) {
        System.out.println(symptom);
    }
    break;
}
return String.join(", ", selectedSymptoms);
};

```

Pour écrire la fonction, il faut suivre les étapes suivantes :

1. Définissez la signature de la fonction en spécifiant le type de retour et les éventuels paramètres nécessaires.
2. Créez une liste vide appelée "selectedSymptoms" pour stocker les symptômes sélectionnés par l'utilisateur.
3. Initialiser un objet Scanner pour lire les entrées de l'utilisateur à partir de la console.
4. Démarrer une boucle "while" qui s'exécute en continu jusqu'à ce que l'utilisateur indique qu'il a terminé la sélection.
5. À l'intérieur de la boucle, affichez le menu des symptômes en utilisant une boucle "for" pour parcourir la liste des symptômes. Numérotez les symptômes pour faciliter la sélection.
6. Lisez l'entrée de l'utilisateur à l'aide de la méthode "nextLine" de l'objet Scanner et stockez-la dans une variable.
7. Analysez l'entrée de l'utilisateur en utilisant la méthode "split" pour séparer les choix de symptômes entrés par l'utilisateur. Les choix peuvent être séparés par des virgules, des espaces ou d'autres caractères définis.
8. Parcourez les choix obtenus à l'étape précédente à l'aide d'une boucle "for". À l'intérieur de la boucle, convertissez chaque choix en un entier en utilisant la méthode "parseInt" de la classe Integer.

9. Vérifiez si l'index obtenu est valide, c'est-à-dire s'il est compris entre 1 et la taille de la liste des symptômes. Si l'index est valide, récupérez le symptôme correspondant à l'index dans la liste des symptômes.
10. Vérifiez si le symptôme récupéré n'est pas déjà présent dans la liste des symptômes sélectionnés. Si le symptôme n'est pas présent, ajoutez-le à la liste des symptômes sélectionnés.
11. Gérez les cas où l'index est invalide ou si la conversion de l'entrée en entier échoue. Affichez des messages d'erreur appropriés pour informer l'utilisateur.
12. Demandez à l'utilisateur s'il a terminé la sélection en affichant un message approprié.
13. Lisez l'entrée de l'utilisateur et stockez-la dans une variable.
14. Vérifiez si l'entrée indique que l'utilisateur a terminé la sélection. Si oui, affichez les symptômes sélectionnés à l'écran et sortez de la boucle "while".
15. Retournez les symptômes sélectionnés en les concaténant avec des virgules ou tout autre séparateur souhaité.
16. Finalisez la fonction en ajoutant les éventuelles gestion d'exceptions et des messages de sortie pour guider l'utilisateur

- Classe interne RequestAppointmentBehaviour :

Cette classe est une sous-classe de *CyclicBehaviour*, qui est un comportement cyclique de Jade. Le comportement de cet agent consiste à recevoir et à traiter les messages reçus.

Selon le type de message reçu, différentes actions sont effectuées, telles que l'envoi de réponses, l'affichage de messages à la console, etc. Vous retrouverez le code complet de cette méthode en annexe.

```
private class RequestAppointmentBehaviour extends CyclicBehaviour {
    public void action() {
        ACLMessage response = receive();
        if (response != null) {
            switch (response.getConversationId()) {
                case "appointment-confirmation": ...
                case "appointment-waiting": ...
                case "call-appointment": ...
                case "Doctor": ...
                case "patientSymptoms": ...
                case "prescription": ...
                case "goodbye-response": ...
                case "notebook": ...
                default:
                    // Handle other cases here
            }
        }
    }
}
```

A titre de guide

ID de conversation	Expéditeur	Type de message	Type de réponse	ID de réponse du patient
Notebook	Secrétaire	REQUEST	INFORM	
goodbyereponse	Secrétaire ou Médecin	INFORM		
prescription	Médecin	INFORM	INFORM	goodbye-from-patient
patientSymptoms	Médecin	REQUEST	INFORM	symptoms
Doctor	Médecin	INFORM	INFORM	patient
Call-appointment	Secrétaire	INFORM	INFORM	
appointment-waiting	Secrétaire	PROPOSAL	CONFIRM ou REJECT_PROPOSAL	goodbye-from-patient ou accept-proposal-waiting-by-patient
appointment-confirmation	Secrétaire	INFORM		

- **Notebook** : Demande du carnet médical du patient
- **prescription** : Contient la prescription médicale pour le patient
- **patientSymptoms** : Demande au patient d'indiquer ses symptômes
- **doctor**: Indique au patient qu'il discute avec le médecin
- **call-appointment** : Demande au patient de prendre rendez-vous chez le médecin
- **appointment-waiting** : Demande au patient s'il peut attendre car le médecin est occupé.
- **appointment-confirmation** : Confirme la programmation du rendez-vous entre le patient et le médecin
- **symptoms** : Indique au médecin que le message contient la liste des symptômes du patient
- **patient** : Indique au médecin le patient dont il s'occupe

6.3.3 Agent Secretaire

6.3.3.1 *SecretaryContainer*

La classe SecretaryContainer est responsable de la configuration et du démarrage du conteneur d'agents pour l'agent "Secrétaire" dans le système Jade. Il suffit de prendre le code de container fourni précédemment et d'adapter pour le lancement de la secrétaire :

```
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;

public class SecretaryContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Profile profile = new ProfileImpl( isMain: false );
        profile.setParameter(Profile.MAIN_HOST, s1: "localhost");
        AgentContainer agentContainer = runtime.createAgentContainer(profile);
        AgentController agentController = agentContainer.createNewAgent( nickname: "infirmière", className: "Secretary", new Object[]{});
        agentController.start();
    }
}
```

6.3.3.2 SecretaryAgent

Dans le fichier SecretaryAgent, cet agent agit en tant que secrétaire chargé de gérer les demandes de rendez-vous des patients et de communiquer avec les médecins.

- Initialisation de l'agent :
 - Le code commence par initialiser l'agent "SecretaryAgent" en étendant la classe jade.core.Agent.
 - Les LinkedHashMaps sont créés pour stocker les rendez-vous planifiés et les carnets de rendez-vous des patients.
 - La méthode setup() est utilisée pour configurer l'agent lors de son démarrage.
- Comportement de gestion des rendez-vous :
 - Le comportement *HandleAppointmentRequestBehaviour* est défini pour gérer les différentes actions liées aux rendez-vous.
 - Le comportement est exécuté de manière cyclique en utilisant la méthode *action()*.
 - Le comportement traite les messages reçus par l'agent et effectue des actions en fonction du contenu du message.
- **Définition des méthodes utilitaires :** Dans ce tableau nous avons un ensemble de toutes les méthodes que devra implémenter l'agent SECRETARY

Nous allons maintenant procéder au recensement des méthodes que devra implémenter l'agent Secretary :

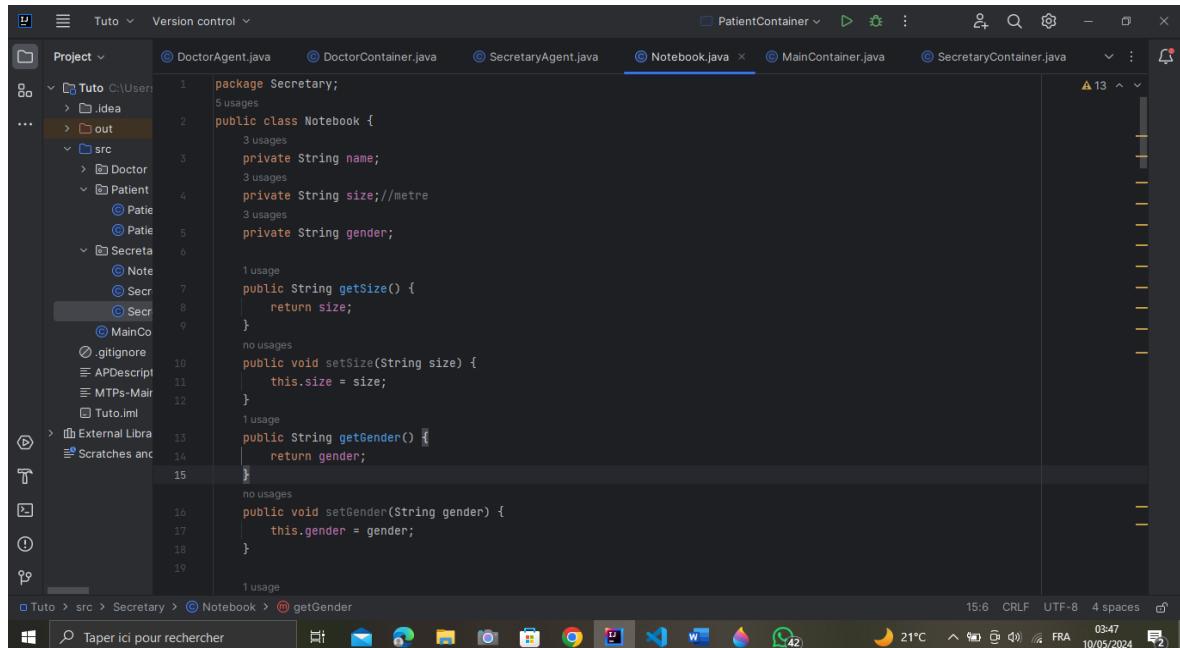
MÉTHODE	DESCRIPTION
TakeNoteBook(AID patient)	Elle envoie une demande au patient pour obtenir son cahier. A l'aide d'objet ACLMessage de type ACLMessage.REQUEST.
NotebookCreation(ACLMessage request)	Elle crée un objet Notebook à partir du contenu du message reçu. A l'aide d'objet

	ACLMensaje de tipo ACLMensaje.REQUEST.
callPatient(AID patient, AID doctor)	Elle envoie une demande au patient pour le rappeler pour le rendez-vous.A l'aide d'objet ACLMensaje de tipo ACLMensaje.REQUEST.
informDoctor(AID doctor, AID patient, Notebook note)	Elle informe le médecin de l'arrivée du patient et envoie le cahier du patient.A l'aide d'objet ACLMensaje de tipo ACLMensaje.INFORM.
informPatientToWait(AID patient)	Elle envoie un message de proposition au patient pour l'informer qu'aucun médecin n'est disponible et qu'il doit attendre.A l'aide d'objet ACLMensaje de tipo ACLMensaje.INFORM.
sendGoodbyeResponse(AID patient)	La méthode envoie une réponse d'au revoir au patient.

6.3.3.3 Class Notebook

Cette classe est créée car elle est un moyen pratique de stocker et de manipuler les informations médicales d'un patient dans un format structuré. En convertissant le carnet de notes en JSON, il devient plus facile de partager et de traiter ces informations avec d'autres systèmes ou services.

- Déclaration des attributs et des getters/setters :

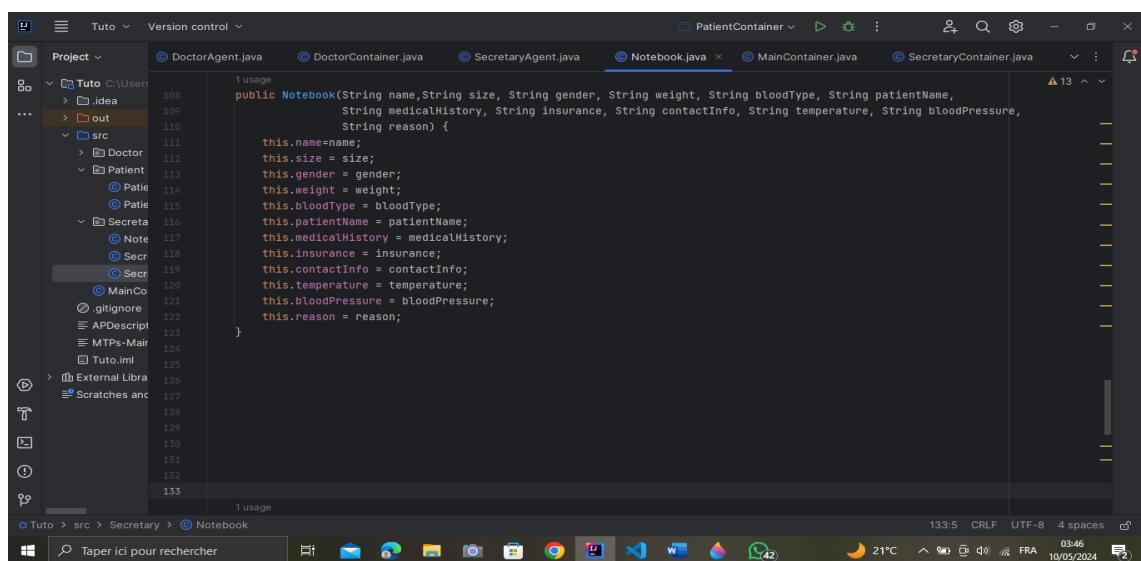


The screenshot shows a Java code editor in a dark-themed IDE. The current file is `Notebook.java`. The code defines a class `Notebook` with three private attributes: `name`, `size`, and `gender`. It includes two getters: `getSize()` and `getGender()`, and two setters: `setSize(String size)` and `setGender(String gender)`.

```
1 package Secretary;
2
3 public class Notebook {
4     private String name;
5     private String size; // metre
6     private String gender;
7
8     public String getSize() {
9         return size;
10    }
11
12    public void setSize(String size) {
13        this.size = size;
14    }
15
16    public String getGender() {
17        return gender;
18    }
19
20    public void setGender(String gender) {
21        this.gender = gender;
22    }
23}
```

Figure 22 Declaration des getters et des setters

- Constructeur de la classe Notebook :



The screenshot shows the same Java IDE environment. The current file is still `Notebook.java`. The code now includes a constructor that takes nine parameters: `String name`, `String size`, `String gender`, `String weight`, `String bloodType`, `String patientName`, `String medicalHistory`, `String insurance`, and `String contactInfo`. The constructor initializes these variables with the provided values.

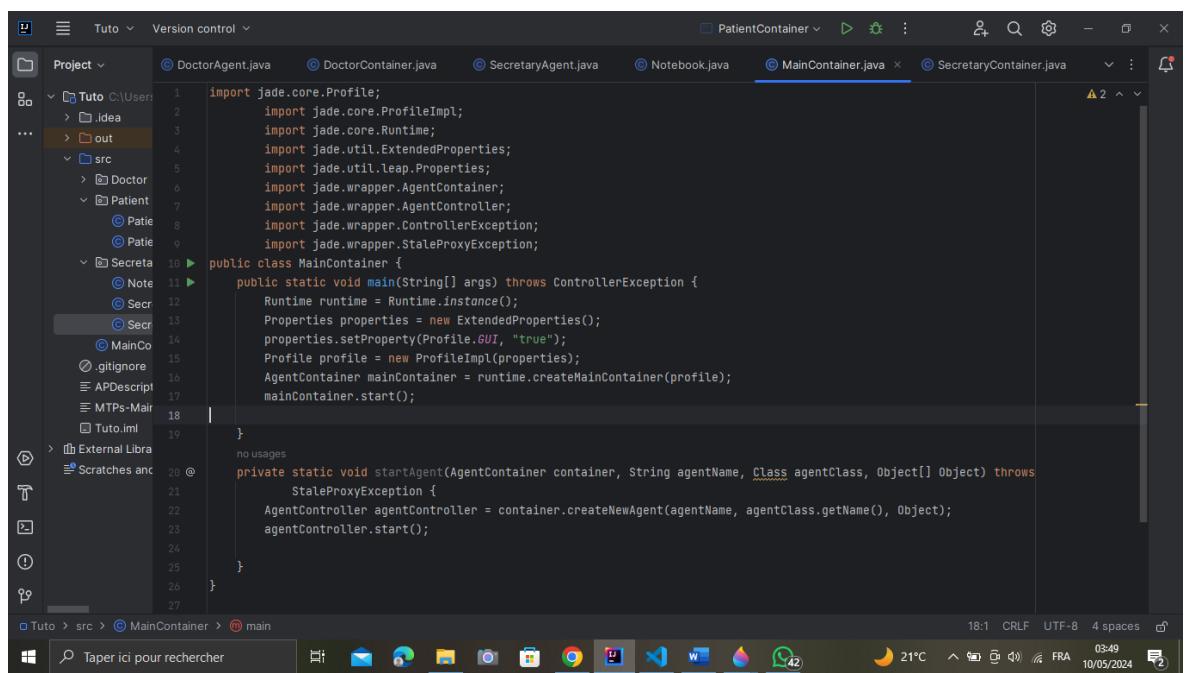
```
1 usage
2 public Notebook(String name, String size, String gender, String weight, String bloodType, String patientName,
3                  String medicalHistory, String insurance, String contactInfo, String temperature, String bloodPressure,
4                  String reason) {
5     this.name=name;
6     this.size = size;
7     this.gender = gender;
8     this.weight = weight;
9     this.bloodType = bloodType;
10    this.patientName = patientName;
11    this.medicalhistory = medicalhistory;
12    this.insurance = insurance;
13    this.contactInfo = contactInfo;
14    this.temperature = temperature;
15    this.bloodPressure = bloodPressure;
16    this.reason = reason;
17}
```

Figure 23 Constructeurs de la classe Notebook

- **Méthode convertNotebookToJson()** : Cette méthode convertit les informations du carnet de notes en une représentation JSON. Elle construit une chaîne de caractères JSON en utilisant les informations du carnet de notes formatées dans un format clé-valeur. La méthode retourne ensuite cette chaîne JSON.

6.3.4 Main Container

Ici nous avons le code qui permet donc de configurer et de démarrer notre conteneur principal dans JADE, prêt à exécuter des agents. Implémenté toujours avec la même logique que les autres conteneurs à la seule différence qu'il est le principal grâce à la ligne 16 du code où on dit *createMainContainer(profile)*:



```

import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.util.ExtendedProperties;
import jade.util.leap.Properties;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;

public class MainContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Properties properties = new ExtendedProperties();
        properties.setProperty(Profile.GUI, "true");
        Profile profile = new ProfileImpl(properties);
        AgentContainer mainContainer = runtime.createMainContainer(profile);
        mainContainer.start();
    }

    private static void startAgent(AgentContainer container, String agentName, Class agentClass, Object[] Object) throws
            StaleProxyException {
        AgentController agentController = container.createNewAgent(agentName, agentClass.getName(), Object);
        agentController.start();
    }
}

```

Figure 24 Main Container class

6.4 EXÉCUTION

La partie code étant terminée voici comment se déroule l'exécution de notre système multi agent

Nous allons suivre un ordre particulier pour l'exécution de chaque Main Container

1. En premier nous exécuterons le Main_Container principal comme présenté ci-dessous:

The screenshot shows an IDE interface with the following details:

- Project View:** Shows a project structure with packages like Tuto, Doctor, Patient, Secretary, Notebook, and MainContainer.
- Code Editor:** Displays the `MainContainer.java` file. The cursor is positioned at line 11, which contains the method signature `private static void main(String[] args)`. A tooltip for this line shows the option "Run 'MainContainer.main()'" with a keyboard shortcut "Ctrl+Maj+F10".
- Status Bar:** Shows the current time as 2:38, file encoding as CRLF, character set as UTF-8, and code style as 4 spaces.
- System Tray:** Shows icons for battery level (20%), weather (20°C Eclairies), signal strength, and date/time (13/05/2024).

Figure 25 Exécution MainContainer.java

Une nouvelle fenêtre s'affiche, il s'agit de l'interface JADE Remote Agent Management GUI. Sur le terminal, on a : *Agent container Main-Container@192.168.56.1 is ready*. Cela indique que le conteneur d'agent avec le nom "Main-Container" est prêt et fonctionne sur l'adresse IP "192.168.56.1".

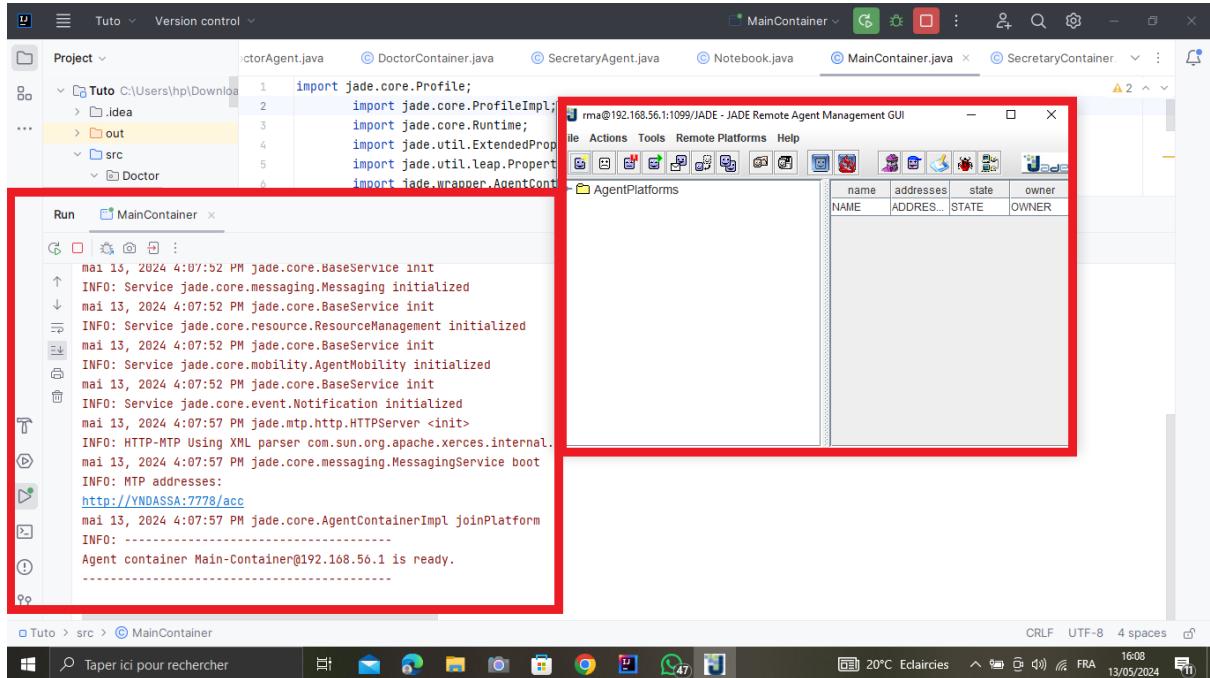


Figure 26 Exécution MainContainer.java 2

2. Sur l'interface JADE Remote Agent Management GUI, se diriger sur l'onglet *Tools* puis sur *Start Sniffer* qui permet de capturer et d'analyser le trafic des messages échangés entre les agents.

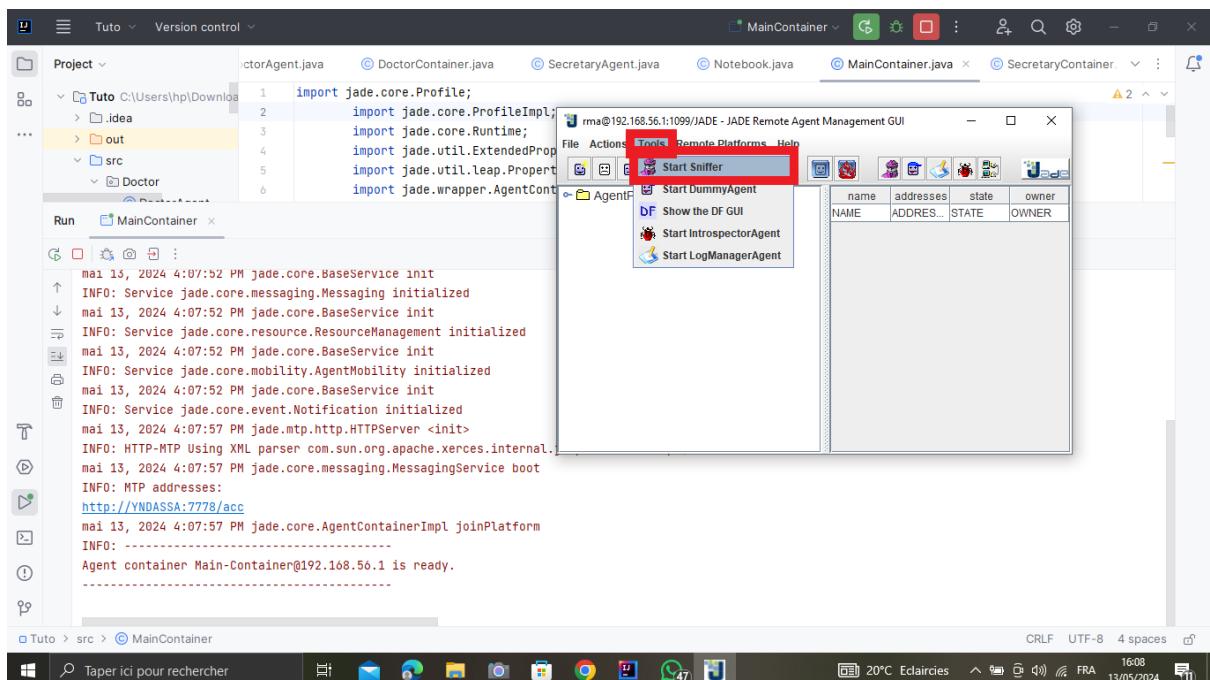


Figure 27 Exécution MainContainer.java 3

Il s'affiche cette interface

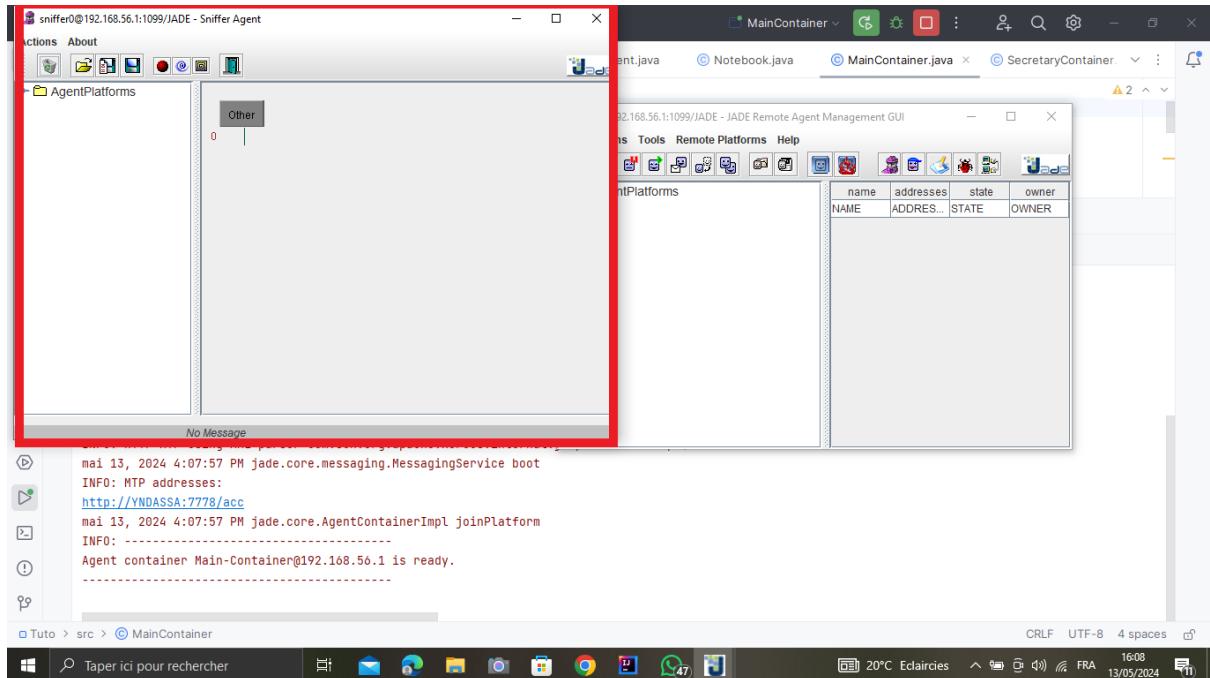


Figure 28 Exécution MainContainer.java 4

3. Revenons dans notre répertoire et exécutons *SecretaryContainer* :

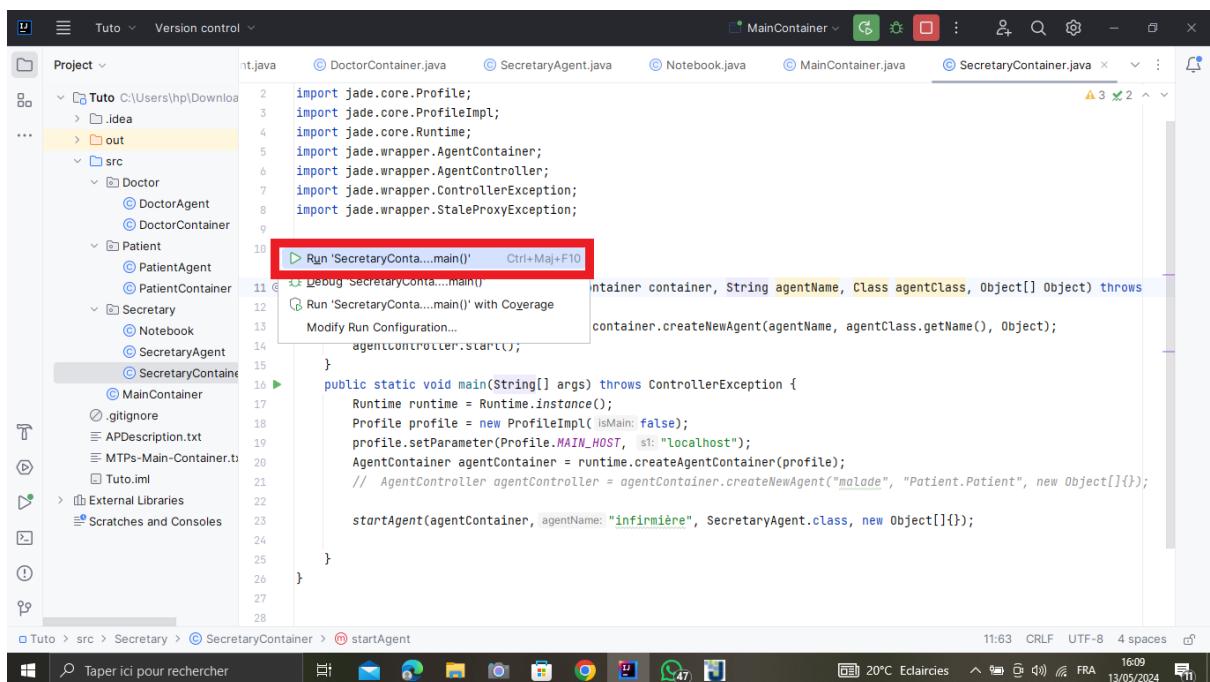


Figure 29 Exécution SecretaryContainer

Vérifier qu'il s'affiche le message encadré en rouge confirmant que l'agent a bel et bien été lancé

```

mai 13, 2024 4:09:08 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
mai 13, 2024 4:09:08 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
mai 13, 2024 4:09:08 PM jade.core.ResourceManagement initialized
mai 13, 2024 4:09:08 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
mai 13, 2024 4:09:08 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
mai 13, 2024 4:09:08 PM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-1@192.168.56.1 is ready.

Agent secretaire infirmiere@192.168.56.1:1099/JADE est prêt

```

Figure 30 Exécution SecretaryContainer 2

4. Revenons dans notre code et executons DoctorContainer

```

import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;

public static void startAgent(AgentContainer container, String agentName, Class agentClass, Object[] object) throws
StaleProxyException {
    AgentController agentController = container.createNewAgent(agentName, agentClass.getName(), object);
    agentController.start();
}

public static void main(String[] args) throws ControllerException {
    Runtime runtime = Runtime.getRuntime();
    Profile profile = new ProfileImpl(isMain: false);
    profile.setParameter(Profile.MAIN_HOST, st: "localhost");
    AgentContainer agentContainer = runtime.createAgentContainer(profile);
    // AgentController agentController = agentContainer.createNewAgent("malade", "Patient.Patient", new Object[]{});
    startAgent(agentContainer, agentName: "medecin1", DoctorAgent.class, new Object[]{});
}

```

Figure 31 Exécution DoctorContainer

Resultat:

```

INFO: Listening for intra-platform commands on address:
- jicp://192.168.56.1:5775

mai 13, 2024 4:09:29 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
mai 13, 2024 4:09:29 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
mai 13, 2024 4:09:29 PM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
mai 13, 2024 4:09:29 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
mai 13, 2024 4:09:29 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
mai 13, 2024 4:09:29 PM jade.core.AgentContainerImpl joinPlatform
INFO:
-----
Agent container Container-2@192.168.56.1 is ready.
-----
```

Figure 32 Exécution DoctorContainer 2

5. Pour finir nous exécuterons PatientContainer

```

import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;

public static void main(String[] args) throws ControllerException {
    Runtime runtime = Runtime.getRuntime();
    Profile profile = new ProfileImpl(isMain: false);
    profile.setParameter(Profile.MAIN_HOST, s: "localhost");
    AgentController agentContainer = runtime.createAgentContainer(profile);

    startAgent(agentContainer, agentName: "malade", PatientAgent.class, new Object[]{});
```

Figure 33 Exécution PatientContainer

6. Nous nous rendons à nouveau sur l'interface JADE Remote Agent Management GUI sur chaque conteneur représenté, cliquer sur *Do sniff this agent(s)*

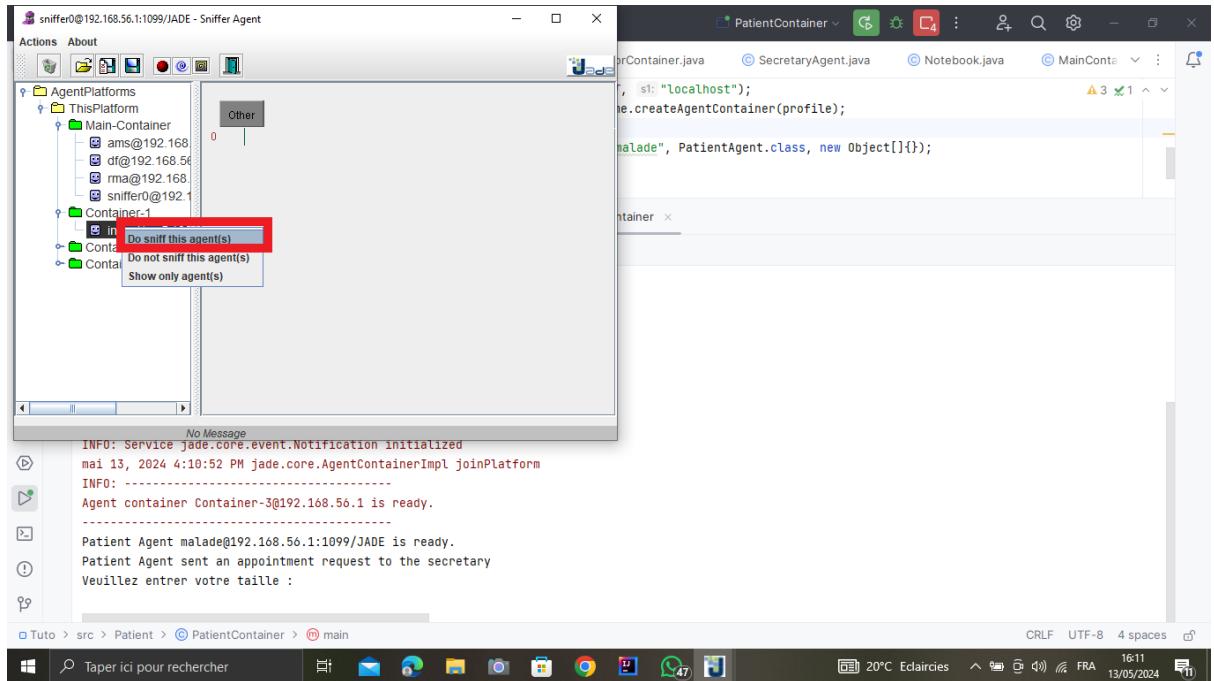


Figure 34 Exécution PatientContainer 2

Résultat:

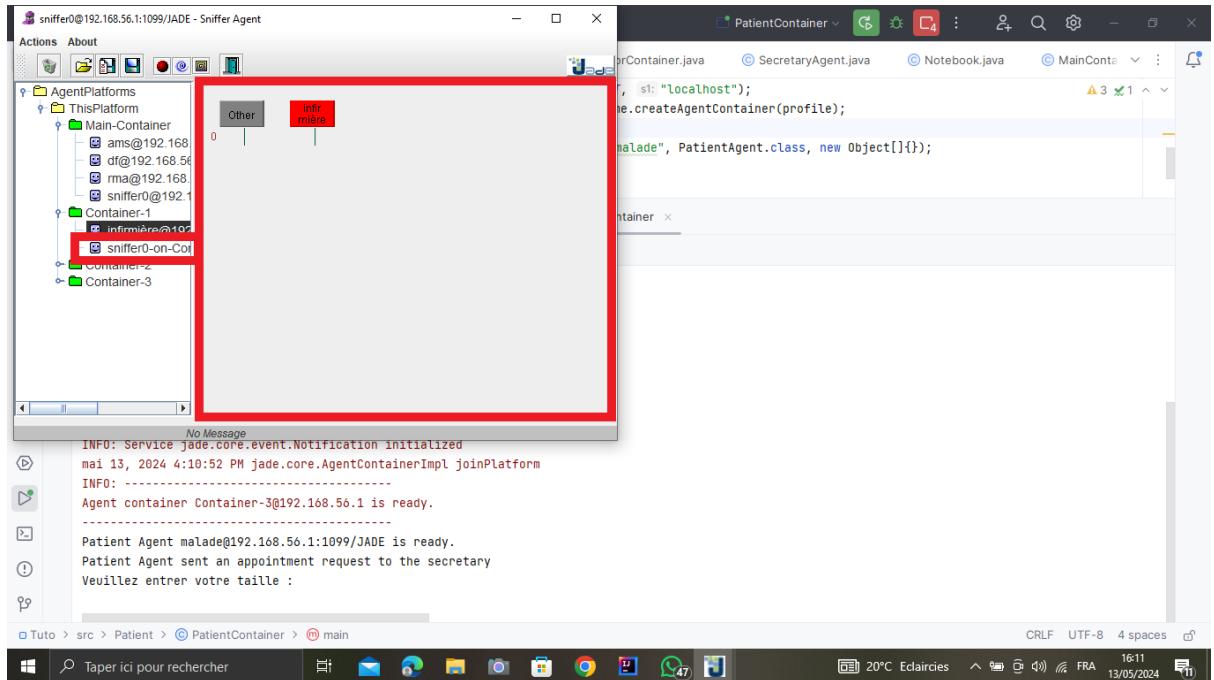


Figure 35 Exécution PatientContainer 3

Continuer le processus pour les autres conteneurs :

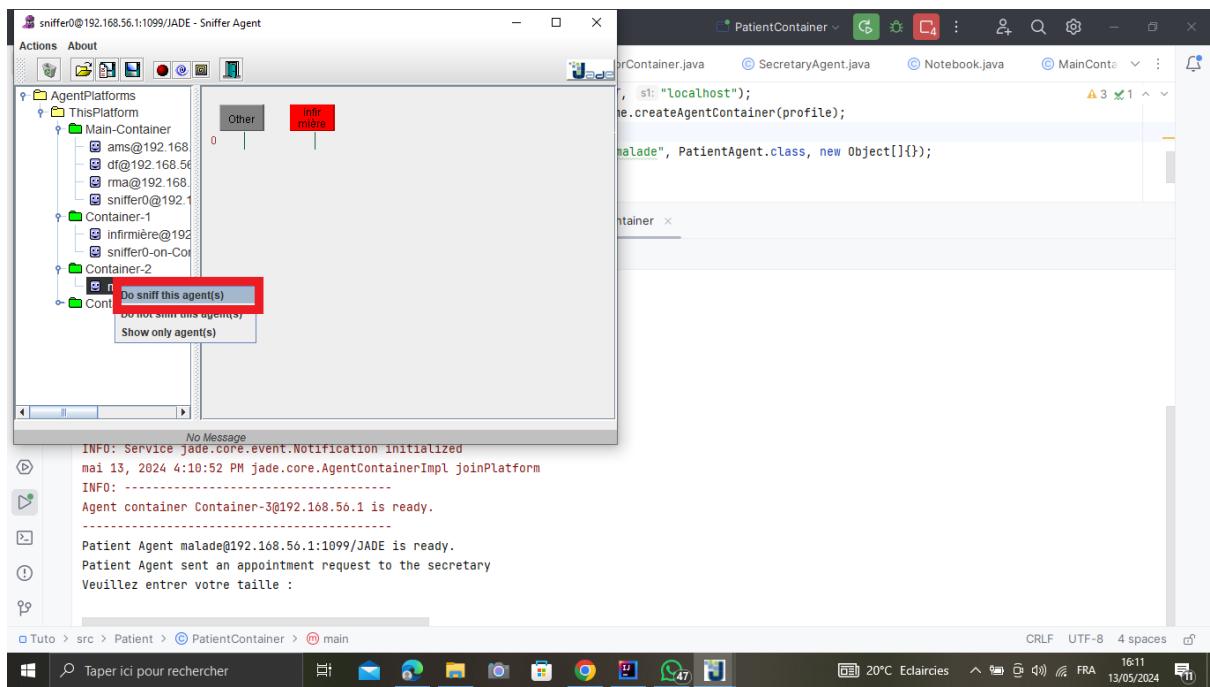


Figure 36 Exécution PatientContainer 4

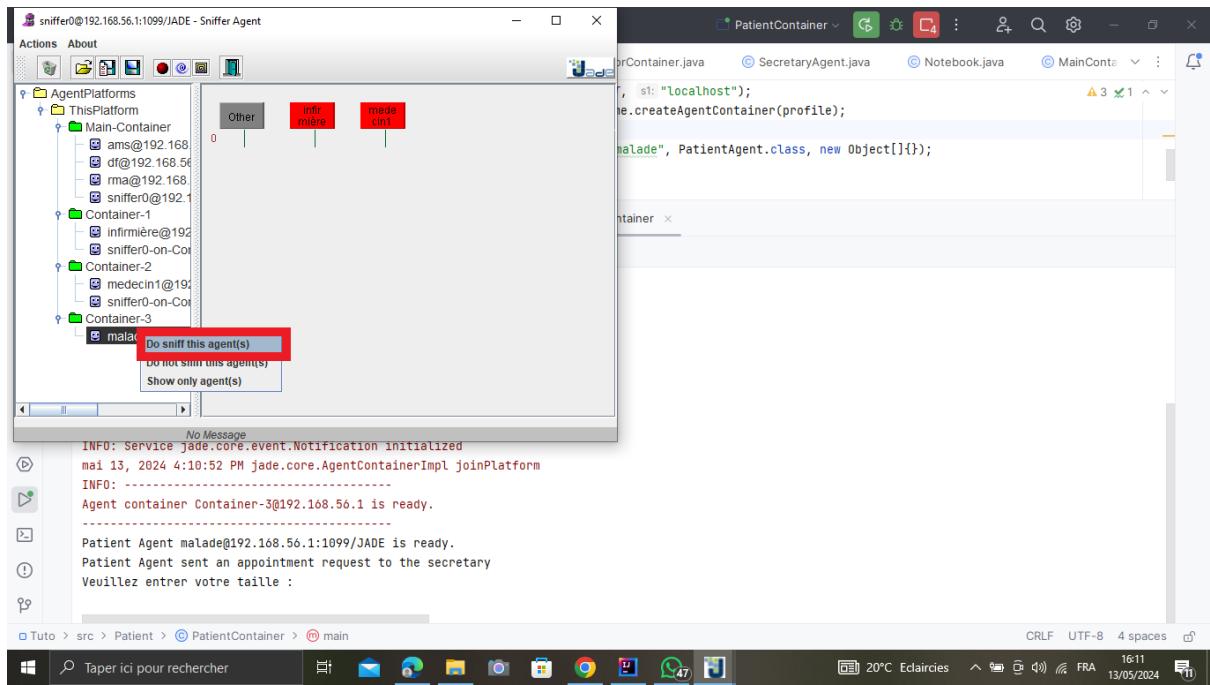


Figure 37 Exécution PatientContainer 5

Et pour terminer Nous avons Ici presenter tous les Conteneur lancés au depart

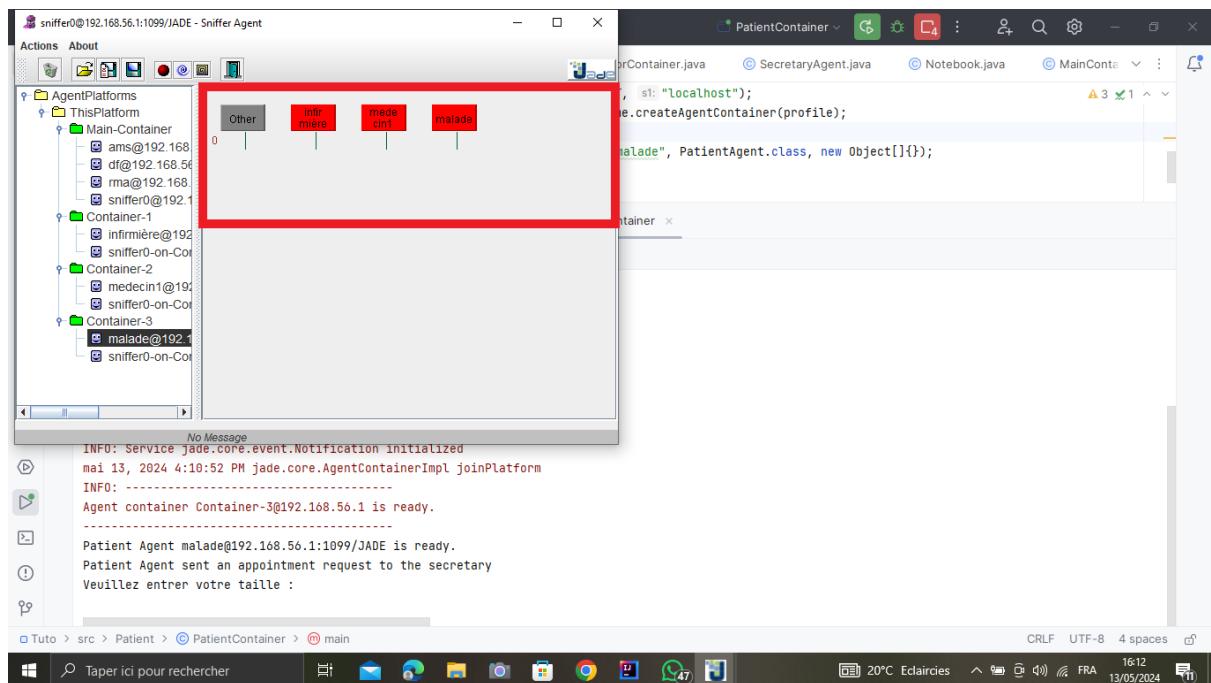


Figure 38 Exécution PatientContainer 6

7. Suivre les instructions sur le terminal. Il s'agit de donner les informations du patient

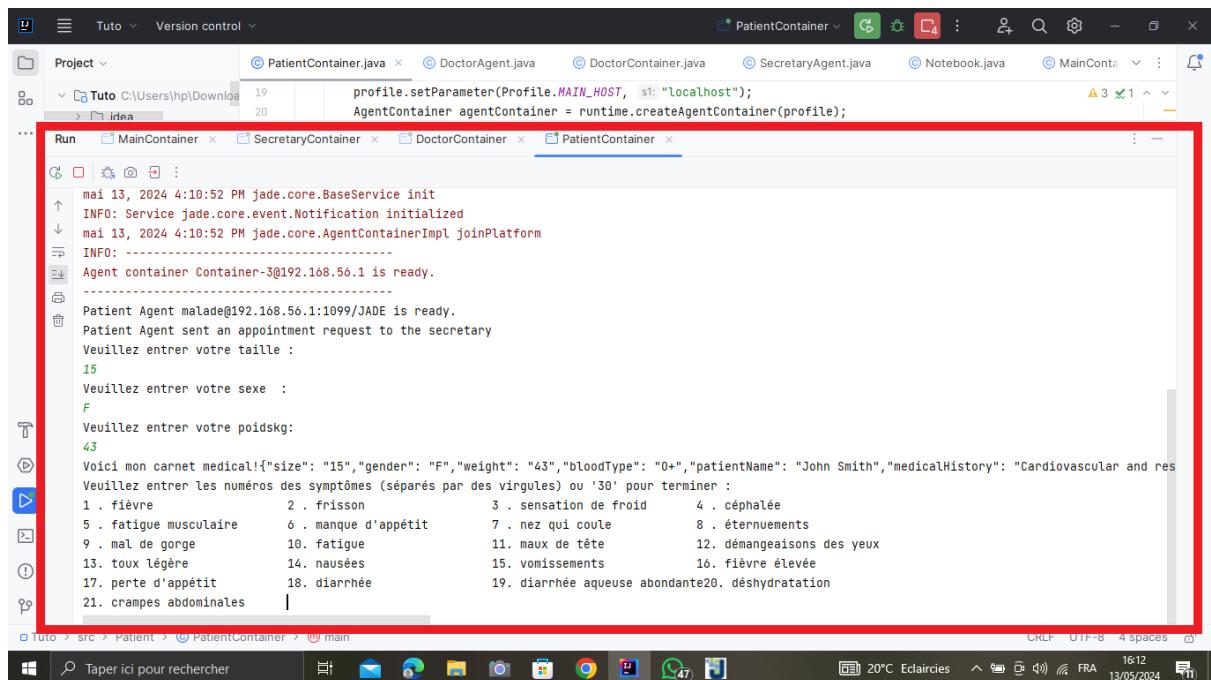


Figure 39 Exécution PatientContainer 7

```

mai 13, 2024 4:10:52 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
mai 13, 2024 4:10:52 PM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Container-3@192.168.56.1 is ready.

Patient Agent malade@192.168.56.1:1099/JADE is ready.
Patient Agent sent an appointment request to the secretary
Veuillez entrer votre taille :
15
Veuillez entrer votre sexe :
F
Veuillez entrer votre poidskg:
43
Voici mon carnet medical!{"size": "15","gender": "F","weight": "43","bloodType": "0+","patientName": "John Smith","medicalHistory": "Cardiovascular and res
Veuillez entrer les numéros des symptômes (séparés par des virgules) ou '30' pour terminer :
1. fièvre          2. frisson          3. sensation de froid          4. céphalée
5. fatigue musculaire 6. manque d'appétit 7. nez qui coule          8. éternuements
9. mal de gorge    10. fatigue         11. maux de tête          12. démangeaisons des yeux
13. toux légère    14. nausées        15. vomissements        16. fièvre élevée
17. perte d'appétit 18. diarrhée       19. diarrhée aqueuse abondante 20. déshydratation
21. crampes abdominales 1,2,4,5,30

```

Figure 40 Exécution PatientContainer 8

8. Après avoir rempli tout sur le terminal, nous pouvons nous rendre dans l'interface JADE Remote Agent Management GUI et voir toutes les interactions qui se sont passées durant les interactions entre les agents Doctor, Secretary et Patient.

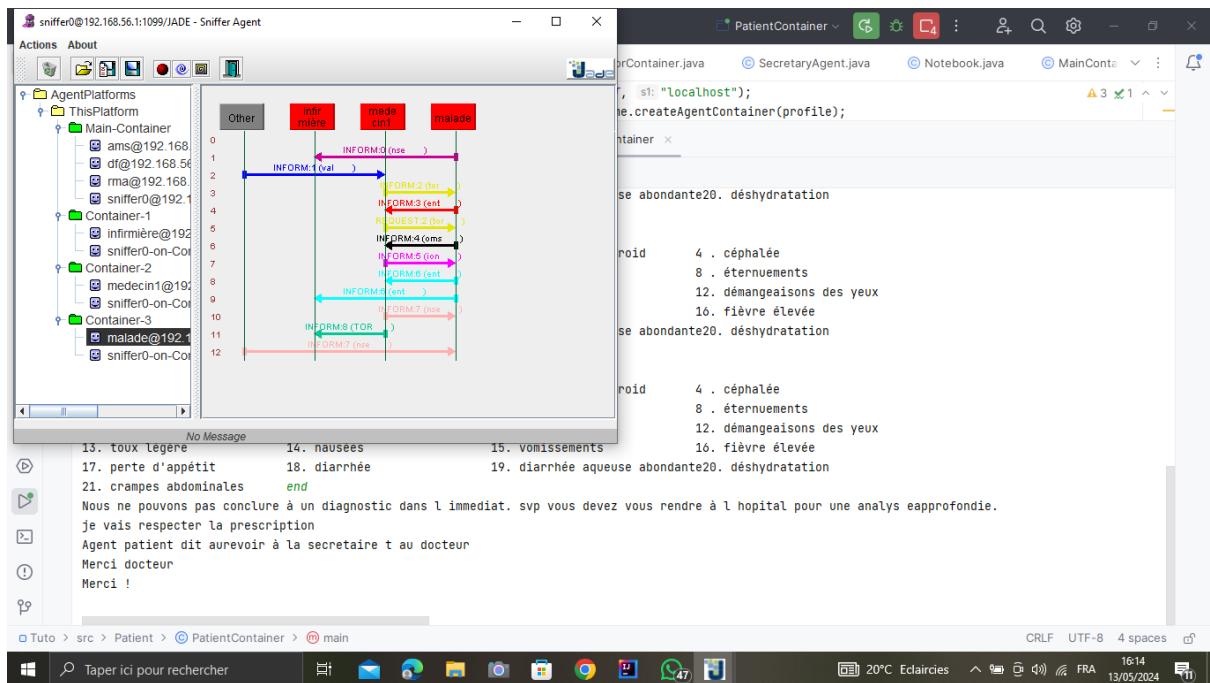


Figure 41 Exécution PatientContainer 9

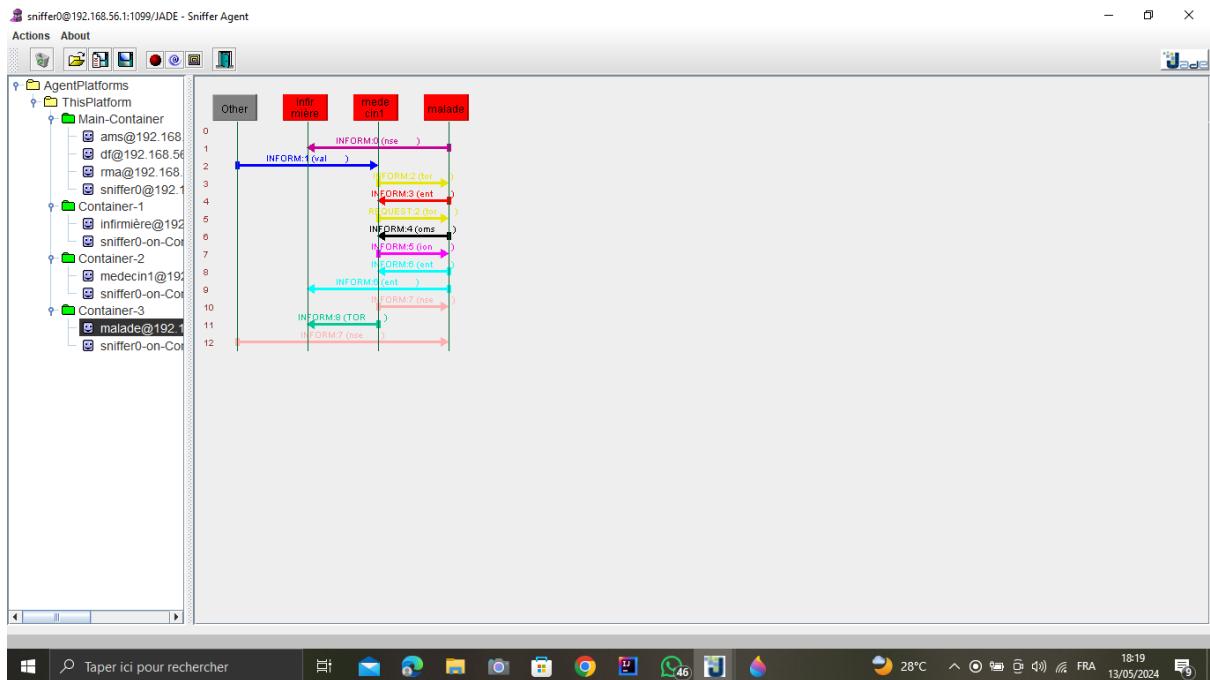


Figure 42 Exécution PatientContainer 10

A la fin nous avons :

```

PatientContainer.java
profile.setParameter(Profile.MAIN_HOST, "localhost");
AgentContainer agentContainer = runtime.createAgentContainer(profile);

// User input handling
17. perte d'appétit      18. diarrhée          19. diarrhée aqueuse abondante 20. déshydratation
21. crampes abdominales  1, 2, 4, 5, 30
Numéro de symptôme invalide : 30
1 . fièvre              2 . frisson           3 . sensation de froid       4 . céphalée
5 . fatigue musculaire   6 . manque d'appétit    7 . nez qui coule            8 . éternuements
9 . mal de gorge         10. fatigue          11. maux de tête             12. démangeaisons des yeux
13. toux légère          14. nausées          15. vomissements            16. fièvre élevée
17. perte d'appétit      18. diarrhée          19. diarrhée aqueuse abondante 20. déshydratation
21. crampes abdominales  30
Numéro de symptôme invalide : 30
1 . fièvre              2 . frisson           3 . sensation de froid       4 . céphalée
5 . fatigue musculaire   6 . manque d'appétit    7 . nez qui coule            8 . éternuements
9 . mal de gorge         10. fatigue          11. maux de tête             12. démangeaisons des yeux
13. toux légère          14. nausées          15. vomissements            16. fièvre élevée
17. perte d'appétit      18. diarrhée          19. diarrhée aqueuse abondante 20. déshydratation
21. crampes abdominales  end

Nous ne pouvons pas conclure à un diagnostic dans l immédiat. svp vous devez vous rendre à l hopital pour une analys eapprofondie.
je vais respecter la prescription
Agent patient dit aurevoir à la secrétaire t au docteur
Merci docteur
Merci |
```

Figure 43 Exécution PatientContainer 11

7 CONCLUSION

Au terme de ce tutoriel, nous avons accompli les différentes tâches prévues. Nous avons d'abord abordé les prérequis essentiels, en nous assurant d'avoir les connaissances de base nécessaires ainsi que les outils adéquats. Ensuite, nous avons procédé à l'installation de JADE, en suivant les étapes nécessaires, et nous avons également configuré JADE dans l'environnement IntelliJ. Par la suite, nous avons mis en pratique nos compétences en implémentant un agent dans un cas général, en couvrant les étapes clés de l'implémentation. Il est important de spécifier que notre objectif pour ce tutoriel a été atteint malgré les difficultés rencontrées, en acquérant une compréhension solide des systèmes multi agents et en menant à bien les différentes étapes du processus. En ce qui concerne les perspectives, celles-ci sont prometteuses. Forts de nos connaissances et de notre expérience, nous sommes désormais en mesure d'explorer des cas pratiques plus avancés, d'approfondir notre compréhension des comportements collectifs des agents et d'explorer des domaines tels que la coordination d'agents et la résolution de problèmes distribués. Les systèmes multi agents offrent un vaste champ d'application et nous sommes confiants que nous serons en mesure de poursuivre votre exploration et de relever de nouveaux défis passionnants dans ce domaine en constante évolution.

8 RÉFÉRENCES BIBLIOGRAPHIQUES

[1]- Site officiel de Jade, JAVA Agent DEvelopment Framework, [en ligne], lien internet: <https://jade.tilab.com/>, consulté le: 09/05/2024.

[2]- djug, Créez votre premier agent avec JADE et ECLIPSE , [en ligne], lien internet: <https://djug.developpez.com/java/jade/creation-agent/> , consulté le: 13/05/2024.

9 ANNEXE

Ici vous trouverez tous les codes nécessaires à l'exécution du Système multi agent de cabinet médical.

MainContainer

```
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.util.ExtendedProperties;
import jade.util.leap.Properties;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;

public class MainContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Properties properties = new ExtendedProperties();
        properties.setProperty(Profile.GUI, "true");
        Profile profile = new ProfileImpl(properties);
        AgentContainer mainContainer =
        runtime.createMainContainer(profile);
        mainContainer.start();

    }

    private static void startAgent(AgentContainer container, String
agentName, Class agentClass, Object[] Object) throws StaleProxyException
{
    AgentController agentController =
    container.createNewAgent(agentName, agentClass.getName(), Object);
    agentController.start();

}
}
```

DoctoContainer

```
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;

public class DoctorContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Profile profile = new ProfileImpl(false);
        profile.setParameter(Profile.MAIN_HOST, "localhost");
        AgentContainer agentContainer =
        runtime.createAgentContainer(profile);
        AgentController agentController =
        agentContainer.createNewAgent("medecin1", "Doctor", new Object[] {});
        agentController.start();
    }

    private static void startAgent(AgentContainer container, String
agentName, Class agentClass) throws StaleProxyException {
        AgentController agentController =
        container.createNewAgent(agentName, agentClass.getName(), null);
        agentController.start();
    }
}
```

Doctor

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.Scanner;

public class Doctor extends Agent {

    Scanner sc= new Scanner(System.in);

    List<String> symptoms = Arrays.asList(
        "fièvre", "frisson", "sensation de froid", "céphalée",
        "fatigue musculaire", "manque d'appétit",
        "nez qui coule", "éternuements", "mal de gorge", "fatigue",
        "maux de tête",
        "démangeaisons des yeux", "toux légère", "nausées",
        "vomissements",
        "fièvre élevée", "perte d'appétit", "diarrhée",
        "diarrhée aqueuse abondante", "déshydratation", "crampes
abdominales"
    );
    // Symptômes pour chaque maladie
    private static final List<String> fluSymptoms =
    Arrays.asList("fièvre", "frisson", "sensation de froid", "céphalée",
    "fatigue musculaire", "manque d'appétit");
    private static final List<String> coldSymptoms = Arrays.asList("nez
qui coule", "éternuements", "mal de gorge", "fatigue", "maux de tête");
    private static final List<String> allergiesSymptoms =
    Arrays.asList("éternuements", "nez qui coule", "démangeaisons des yeux",
    "fatigue", "toux légère");
    private static final List<String> malariaSymptoms =
    Arrays.asList("fièvre", "frissons", "maux de tête", "fatigue", "nausées",
    "vomissements");
    private static final List<String> typhoidSymptoms =
    Arrays.asList("fièvre élevée", "maux de tête", "fatigue", "perte
d'appétit", "diarrhée");
    private static final List<String> choleraSymptoms =
    Arrays.asList("diarrhée aqueuse abondante", "vomissements fréquents",
    "déshydratation", "crampes abdominales");
    private AID secretary=new AID("infirmière", AID.ISLOCALNAME);

    private AID patient;

    boolean isPatient=false;
    private void sendGoodbyeResponse(AID patient) {
        ACLMessage response = new ACLMessage(ACLMessage.INFORM);
        response.setConversationId("goodbye-response");
        response.addReceiver(patient);
        response.setContent("Doctor");
        send(response);
    }
    private void informAvailability(boolean bool) {
```

```

ACLMensaje response = new ACLMensaje(ACLMensaje.INFORM);
response.setConversationId("DOCTOR");
response.addReceiver(secretary);
String input;
if(bool) {
    System.out.println("Secrétaire: Dr svp Etes-vous libre?
1.oui, 2.non");
    input= sc.nextLine();
}
else{
    input="1";
}

if(Objects.equals(input, "1")){
    response.setContent("Doctor is available");
}
else{
    response.setContent("Doctor is not available");
}
send(response);

}

private String extractInformation(String content, String key){
String champRecherche = "\\"+key+"\\:";
int indexDebut = content.indexOf(champRecherche);
if (indexDebut == -1) {
    return null; // Champ non trouvé
}

indexDebut += champRecherche.length();
int indexFin = content.indexOf("}", indexDebut);

if (indexFin == -1) {
    indexFin = content.indexOf("}", indexDebut);
}

if (indexFin == -1) {
    return null; // Fin de champ non trouvé
}

return content.substring(indexDebut, indexFin).replace("\\",
"").trim();
}

private String performDiagnosis(List<String> symptoms) {
if (containsAllSymptoms(symptoms, fluSymptoms)) {
    return "Vous pourriez avoir la grippe.";
} else if (containsAllSymptoms(symptoms, coldSymptoms)) {
    return "Vous pourriez avoir un rhume.";
} else if (containsAllSymptoms(symptoms, allergiesSymptoms)) {
    return "Vous pourriez avoir des allergies.";
} else if (containsAllSymptoms(symptoms, malariaSymptoms)) {
    return "Vous pourriez avoir le paludisme.";
} else if (containsAllSymptoms(symptoms, typhoidSymptoms)) {
    return "Vous pourriez avoir la fièvre typhoïde.";
} else if (containsAllSymptoms(symptoms, choleraSymptoms)) {
    return "Vous pourriez avoir le choléra.";
}
}

```

```

        }
    else {
        return "Nous ne pouvons pas conclure à un diagnostic dans l
immediat. svp vous devez vous rendre à l hopital pour une analys
eapprofondie.";
    }
}

private boolean containsAllSymptoms(List<String> symptoms,
List<String> requiredSymptoms) {
    for (String symptom : requiredSymptoms) {
        if (!symptoms.contains(symptom)) {
            return false;
        }
    }
    return true;
}

protected void setup() {
    addBehaviour(new HandleSecretaryMessagesBehaviour());
}

protected void takeDown() {
    // Perform cleanup tasks here
}

private class HandleSecretaryMessagesBehaviour extends
CyclicBehaviour {
    public void action() {
        ACLMessage msg = receive();
        if (msg != null) {
            switch (msg.getConversationId()) {

                case "askAvailability":
                    informAvailability(true);

                case "patient-arrival":
                    if(!isPatient){
                        isPatient=true;
                        // Step 6: Inform the doctor about the
patient's arrival
                        String message="greetings";
                        String
                    patientLocalName=extractInformation(msg.getContent(), "name");
                        System.out.println("salut :
"+patientLocalName);
                        patient=new AID(patientLocalName,
AID.ISLOCALNAME);
                        ACLMessage greetingsPatient = new
ACLMessages (ACLMessages.INFORM);
                        greetingsPatient.setConversationId("Doctor");
                        greetingsPatient.addReceiver(new
AID(patientLocalName, AID.ISLOCALNAME));
                        greetingsPatient.setContent(message);
                        send(greetingsPatient);
                    }
                    break;
            }
        }
    }
}

```

```

        case "goodbye-from-patient":
            if(Objects.equals(patient.getLocalName(), msg.getSender().getLocalName())){
                isPatient=false;
                patient=null;
                System.out.println("Aurevoir "
+msg.getSender() +"! Prennez soin de vous");
                sendGoodbyeResponse(msg.getSender());
                informAvailability(false);
            }
            break;

        case "patient":
            System.out.println("pouvez vous me decrirre vos
symptomes !?");
            //demande de symptoms aux patients
            ACLMessage symtompMessage = new
ACLMessage(ACLMessage.REQUEST);

symtompMessage.setConversationId("patientSymptoms");
            symtompMessage.addReceiver(patient);
            symtompMessage.setContent(String.join(",",
symptoms));
            send(symtompMessage);
            break;

        case "symptoms":
            // Effectuer le diagnostic en fonction des
symptômes du patient
            System.out.println("symptomes notés !");
            List<String> restoredList = Arrays.asList(
msg.getContent().split(","));
            String diagnosis =
performDiagnosis(restoredList);
            System.out.println("voici votre prescription");
            ACLMessage diagnosismsg = new
ACLMessage(ACLMessage.INFORM);
            diagnosismsg.setConversationId("prescription");
            diagnosismsg.addReceiver(patient);
            diagnosismsg.setContent(diagnosis);
            send(diagnosismsg);
            break;

        // Handle other types of messages here if needed
    }
}
}
}

```

SecretaryContainer

```
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;

public class SecretaryContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Profile profile = new ProfileImpl(false);
        profile.setParameter(Profile.MAIN_HOST, "localhost");
        AgentContainer agentContainer =
        runtime.createAgentContainer(profile);
        AgentController agentController =
        agentContainer.createNewAgent("infirmière", "Secretary", new Object[]{} );
        agentController.start();
    }
}
```

Secretary

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;


public class Secretary extends Agent {
    private LinkedHashMap<AID, String> scheduledAppointments = new
    LinkedHashMap<>();
    private LinkedHashMap<AID, Notebook> Notebooks = new
    LinkedHashMap<>();
    private boolean nobody=true;
    // private boolean isAppointmentScheduled = true;
    // In this example, let's assume the appointment is scheduled
    successfully
    // private String appointmentTime = "2024-04-15 07:52 matin";

    @Override
    protected void setup() {
        System.out.println("Agent secrétaire " + getAID().getName() + " est prêt");

        // Add behavior for handling patient's appointment request
        addBehaviour(new HandleAppointmentRequestBehaviour());
    }
    @Override
```

```

protected void takeDown() {
    System.out.println("END LIFE !");
}

private class HandleAppointmentRequestBehaviour extends
CyclicBehaviour {
    public void action() {

        // MessageTemplate template =
MessageTemplate.MatchPerformativ(ACLMessage.REQUEST);
        ACLMessage request = receive();

        if (request != null) {
            switch (request.getConversationId()) {

                case "appointment-from-patient":
                    System.out.println("la secrétaire agent vient de
recevoir une demande de rendez-vous " + request.getSender().getName());
                    //Step 3: Schedule appointment or inform patient
to wait
                    if (!checkHashMapEmpty()) {
                        System.out.println("Actuellement, aucun
médecin n'est disponible. Allez-vous pouvoir attendre ?.");
                        informPatientToWait(request.getSender());

                    } else {
                        System.out.println("Pouvez-vous me donner
votre carnet médical ?.");
                        TakeNoteBook(request.getSender());
                        String status="processing";

scheduledAppointments.put(request.getSender(), status);

                    }
                    break;

                case "goodbye-from-patient":
                    if (request.getPerformative() ==
ACLMessage.REJECT_PROPOSAL) {

                        // System.out.println("Secretary Agent
received a reject from " + request.getSender().getName());

                    } else {
                        // System.out.println("Secretary Agent
received a goodbye message from " + request.getSender().getName());

                    }
                    System.out.println("Aurevoir "
+request.getSender() +"! prenez soin de vous !");
                    sendGoodbyeResponse(request.getSender());
                    break;

                case "accept-proposal-waiting-by-patient" :

                    String status="waiting";
                    scheduledAppointments.put(request.getSender(),
status);
                    nobody=false;
                    System.out.println("Merci M. /MM "+

}

```

```

request.getSender().getLocalName() + " pour votre comprehension !");
System.out.println("pouvez vous me donner votre
carnet medical ");
TakeNoteBook(request.getSender());
break;

case "DOCTOR":
if(request.getContent().equals("Doctor is
available")) {
removeFirstElement();
String newStatus = "processing";
AID nextPatient =
setFirstInsertedElementStatusAndGetAID(newStatus);
callPatient(nextPatient,request.getSender());

}
else if(request.getContent().equals("End")) {
String message="Doctors are no longer
available. Sorry please come later";
sendMessageToWaitingAppointments(message);

}
break;

case "notebook-response":

Notebook notebook=NotebookCreation(request);
Notebooks.put(request.getSender(),notebook);
System.out.println("carnet medical recu. Veuillez
vous rendre chez le medecin!");

if(nobody) {
informDoctor(new AID("medecin1",
AID.ISLOCALNAME),request.getSender(),Notebooks.get(request.getSender()));
nobody=false;
}

break;

case "proceedToAppointment":
if(request.getContent().equals("1") ){
informDoctor(new AID(request.getContent(),
AID.ISLOCALNAME),request.getSender(),Notebooks.get(request.getSender()));
break;

}
else{
sendGoodbyeResponse(request.getSender());

};

break;
}

}

private void TakeNoteBook(AID patient){
ACLMessage req = new ACLMessage(ACLMessage.REQUEST);

```

```

        req.setConversationId("notebook");
        req.addReceiver(patient);
        req.setContent("");
        send(req);

    }

    private Notebook NotebookCreation(ACLMensaje request) {

        String name=request.getSender().getLocalName();

        // Extract the information from the message
        String size = extractInformation(request.getContent(), "size");
        String gender = extractInformation(request.getContent(),
"gender");
        String weight = extractInformation(request.getContent(),
"weight");
        String bloodType = extractInformation(request.getContent(),
"bloodType");
        String patientName = extractInformation(request.getContent(),
"patientName");
        String medicalHistory = extractInformation(request.getContent(),
"medicalHistory");
        String insurance = extractInformation(request.getContent(),
"insurance");
        String contactInfo = extractInformation(request.getContent(),
"contactInfo");
        String temperature = extractInformation(request.getContent(),
"temperature");
        String bloodPressure = extractInformation(request.getContent(),
"bloodPressure");
        String reason = extractInformation(request.getContent(),
"reason");

        return new
Notebook(name,size,gender,weight,bloodType,patientName,medicalHistory,
insurance, contactInfo, temperature,bloodPressure,reason);
    }

    private void callPatient(AID patient,AID doctor) {
        // Step 3: Send a message to the patient
        ACLMensaje notification = new ACLMensaje(ACLMensaje.REQUEST);
        notification.setConversationId("call-appointment");
        notification.addReceiver(patient);
        notification.setContent(doctor.getLocalName());
        send(notification);
    }

    private void informDoctor(AID doctor,AID patient,Notebook note){
        // Step 6: Inform the doctor about the patient's arrival
        ACLMensaje informDoctor = new ACLMensaje(ACLMensaje.INFORM);
        informDoctor.setConversationId("patient-arrival");
        informDoctor.addReceiver(doctor);
        informDoctor.setContent(note.convertNotebookToJson());
        send(informDoctor);
    }

    private void informPatientToWait(AID patient) {
        ACLMensaje notification = new ACLMensaje(ACLMensaje.PROPOSE);
        notification.setConversationId("appointment-waiting");
        notification.addReceiver(patient);
        notification.setContent("Actuellement, le Dr est
indisponible? Pouvez-vous attendre? 1.oui ,2.non.");
    }
}

```

```

        send(notification);
    }

private void sendGoodbyeResponse(AID patient) {
    ACLMessage response = new ACLMessage(ACLMessage.INFORM);
    response.setConversationId("goodbye-response");
    response.addReceiver(patient);
    response.setContent("Secretary");
    send(response);
}

public boolean checkHashMapEmpty() {
    return this.scheduledAppointments.isEmpty();
}

public void removeFirstElement() {
    Iterator<Map.Entry<AID, String>> iterator =
scheduledAppointments.entrySet().iterator();

    if (iterator.hasNext()) {
        iterator.next();
        iterator.remove();

        if (scheduledAppointments.isEmpty()) {
            nobody = true;
        } else {
            //System.out.println("Empty List.");
        }
    }
}

public Map.Entry<AID, String> getFirstInsertedElement() {
    // Récupérer un ensemble de paires clé-valeur de la LinkedHashMap
    Set<Map.Entry<AID, String>> entrySet =
scheduledAppointments.entrySet();

    // Vérifier si la liste n'est pas vide
    if (!entrySet.isEmpty()) {
        // Récupérer le premier élément inséré en utilisant un
        itérateur
        Iterator<Map.Entry<AID, String>> iterator =
entrySet.iterator();
        return iterator.next();
    } else {
        //System.out.println("Empty list");
        return null;
    }
}

public AID setFirstInsertedElementStatusAndGetAID(String newStatus) {
    // Récupérer le premier élément inséré de la LinkedHashMap
    Map.Entry<AID, String> firstInsertedEntry =
getFirstInsertedElement();

    if (firstInsertedEntry != null) {
        // Récupérer l'AID du premier élément inséré
        AID firstInsertedAID = firstInsertedEntry.getKey();

        // Modifier le statut du premier élément inséré
        firstInsertedEntry.setValue(newStatus);
        System.out.println("Status du patient" + firstInsertedAID + " " +
newStatus);
    }
}

```

```

    réussi avec succès");

        return firstInsertedAID;
    } else {
        //System.out.println("Empty list.");
        return null;
    }
}

public void sendMessageToWaitingAppointments( String message) {
    for (Map.Entry<AID, String> entry :
scheduledAppointments.entrySet()) {
        if (entry.getValue().equalsIgnoreCase("waiting")) {
            // Send a message to the entry with a status of "waiting"

            ACLMessage aclMessage = new
ACLMMessage (ACLMMessage.INFORM);
            aclMessage.addReceiver(entry.getKey());
            aclMessage.setContent(message);
            send(aclMessage);
        }
    }
}

private String extractInformation(String content, String key) {
    String champRecherche = "\\"+key+":";
    int indexDebut = content.indexOf(champRecherche);
    if (indexDebut == -1) {
        return null; // Champ non trouvé
    }

    indexDebut += champRecherche.length();
    int indexFin = content.indexOf(",", indexDebut);

    if (indexFin == -1) {
        indexFin = content.indexOf("}", indexDebut);
    }

    if (indexFin == -1) {
        return null; // Fin de champ non trouvé
    }

    return content.substring(indexDebut, indexFin).replace("\\\"", "");
}

public boolean uniqueElement(){
    return scheduledAppointments.size() == 1;
}

public String getDate() {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm
a");
    Date dateActuelle = new Date();
    return sdf.format(dateActuelle);
}

public boolean compareDates(String date1, String date2) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm
a");
    try {
        Date d1 = sdf.parse(date1);

```

```

        Date d2 = sdf.parse(date2);

        return d1.equals(d2);
    } catch (ParseException e) {
        e.printStackTrace();
        return false;
    }
}

private void scheduleAppointment(AID patient, String appointmentTime)
{
    ACLMessage confirmation = new ACLMessage(ACLMessage.CONFIRM);
    confirmation.setConversationId("appointment-confirmation");
    confirmation.addReceiver(patient);
    confirmation.setContent("Your appointment is scheduled for " +
appointmentTime + ". Please be ready at the appointed time.");
    send(confirmation);
}

private void checkDoctorAvailability() {
    // Send availability requests to doctors and wait for their
responses
    ACLMessage availabilityRequest = new
    ACLMessage(ACLMessage.REQUEST);
    availabilityRequest.setConversationId("askAvailability");
    availabilityRequest.setContent("Doctor availability request");
    availabilityRequest.addReceiver(new AID("medecin1",
AID.ISLOCALNAME));
    //availabilityRequest.addReceiver(new AID("medecin2",
AID.ISLOCALNAME));
    send(availabilityRequest);
}
}

```

PatientContainer

```

import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;

public class PatientContainer {
    private static void startAgent(AgentContainer container, String
agentName, Class agentClass, Object[] Object) throws StaleProxyException
{
    AgentController agentController =
container.createNewAgent(agentName, agentClass.getName(), Object);
    agentController.start();
}
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        Profile profile = new ProfileImpl(false);
        profile.setParameter(Profile.MAIN_HOST, "localhost");
        AgentContainer agentContainer =
runtime.createAgentContainer(profile);
        // AgentController agentController =
agentContainer.createNewAgent("malade", "Patient.Patient", new
Object[]{});

        startAgent(agentContainer, "malade", Patient.class, new
Object[]{});
    }
}

```

Patient

```

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

import java.util.*;

public class Patient extends Agent {
    List<String> symptoms;
    AID doctor;
    @Override
    protected void setup() {
        System.out.println("Patient Agent " + getAID().getName() + " is
ready.");
    }

    // Step 1: Send an appointment request to the secretary
    ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
    request.setConversationId("appointment-from-patient");
    request.setContent("Appointment request");

```

```

        request.addReceiver(new AID("infirmière", AID.ISLOCALNAME));
        send(request);
        System.out.println("Le Patient a envoyé une demande de rendez-
vous chez à l'infirmière");

        // Add behavior for requesting an appointment
        addBehaviour(new RequestAppointmentBehaviour());
    }

    @Override
    protected void takeDown() {
        System.out.println("END LIFE !");
    }

    private String Notebook() {

        Scanner scanner = new Scanner(System.in);

        // Variables pour les informations du patient
        System.out.println("Veuillez entrer votre taille :");
        String size = scanner.nextLine();

        System.out.println("Veuillez entrer votre sexe :");
        String gender = scanner.nextLine();

        System.out.println("Veuillez entrer votre poids en kg:");
        String weight = scanner.nextLine();
        // Variables pour les du patient
        //String size = "175";
        //String gender = "Male";
        //String weight = "70";
        String bloodType = "O+";
        String patientName = "John Smith";
        String medicalHistory = "Cardiovascular and respiratory
conditions";
        String insurance = "Yes";
        String contactInfo = "john@example.com";
        String temperature = "37.5";
        String bloodPressure = "120/80";
        String reason = "Routine check-up";

        // Construire le contenu du message en utilisant la concaténation
        // de chaînes
        String content = "{"
            + "\"size\": \"\" + size + "\","
            + "\"gender\": \"\" + gender + "\","
            + "\"weight\": \"\" + weight + "\","
            + "\"bloodType\": \"\" + bloodType + "\","
            + "\"patientName\": \"\" + patientName + "\","
            + "\"medicalHistory\": \"\" + medicalHistory + "\","
            + "\"insurance\": \"\" + insurance + "\","
            + "\"contactInfo\": \"\" + contactInfo + "\","
            + "\"temperature\": \"\" + temperature + "\","
            + "\"bloodPressure\": \"\" + bloodPressure + "\","
            + "\"reason\": \"\" + reason + \"\""
            + "}";

        return content;
    }

    private String getSymptoms() {

```

```

List<String> selectedSymptoms = new ArrayList<>();

Scanner scanner = new Scanner(System.in);

while (true) {
    // Affichage du menu
    int counter = 1;
    for (String symptom : symptoms) {
        System.out.printf("%-2d. %-25s", counter, symptom);
        if (counter % 4 == 0) {
            System.out.println();
        }
        counter++;
    }
    System.out.println();
    // Lecture des choix du patient
    String input = scanner.nextLine();

    // Analyse des choix du patient
    String[] choices = input.split(",");
    for (String choice : choices) {
        try {
            int index = Integer.parseInt(choice.trim());
            if ((index >= 1 && index <= symptoms.size()) ||
index>=30) {
                String symptom = symptoms.get(index - 1);
                if (!selectedSymptoms.contains(symptom)) {
                    selectedSymptoms.add(symptom);
                }
            } else {
                System.out.println("Numéro de symptôme invalide : "
" + index);
            }
        } catch (NumberFormatException e) {
            System.out.println("Format de numéro invalide : " +
choice);
        }
    }

    System.out.println("Avez-vous terminé? 1. oui ,2. non");
    String entry = scanner.nextLine();
    if (entry.equals("1")) {
        System.out.println("Symptômes sélectionnés :");
        for (String symptom : selectedSymptoms) {
            System.out.println(symptom);
        }
        break;
    }
}

return String.join(", ", selectedSymptoms);
};

private class RequestAppointmentBehaviour extends CyclicBehaviour {

```

```

public void action() {
    ACLMessage response = receive();
    if (response != null) {
        switch (response.getConversationId()) {
            case "appointment-confirmation":
                System.out.println("Patient Agent a recu une confirmation de rdv de la secretaire");
                System.out.println(response.getContent());
                break;

            case "appointment-waiting":
                //logique secretaire
                System.out.println("Secretaire : " +
response.getContent());
                Scanner scanner = new Scanner(System.in);
                String input = scanner.nextLine();
                ACLMessage notification;
                if(Objects.equals(input, "1"))
                {
                    notification = new
ACLMensaje (ACLMensaje.CONFIRM);
                    notification.setConversationId("accept-
proposal-waiting-by-patient");

                }
                else{
                    notification = new
ACLMensaje (ACLMensaje.REJECT_PROPOSAL);
                    notification.setConversationId("goodbye-from-
patient");
                }
                notification.addReceiver(response.getSender());
                notification.setContent(input);
                scanner.close();
                send(notification);

                break;

            case "call-appointment":

                System.out.println("Secretaire : " + " Vous y allez Mr ? 1.Oui , 2.non :");
                Scanner scanners = new Scanner(System.in);
                String inputs = scanners.nextLine();
                ACLMessage decision = new
ACLMensaje (ACLMensaje.INFORM);

                decision.setConversationId("proceedToAppointment");
                decision.addReceiver(response.getSender());
                doctor=new AID(response.getContent(),
AID.ISLOCALNAME);
                decision.setContent(inputs);
                scanners.close();
                send(decision);

                break;

            case "Doctor":
                if(Objects.equals(response.getContent(),

```

```

"greetings")) {
    String patientResponse = "greetings";

    ACLMessage answer = new
ACLMessage(ACLMessage.INFORM);
    answer.addReceiver(response.getSender());
    answer.setConversationId("patient");
    answer.setContent(patientResponse);
    send(answer);
}
break;

case "patientSymptoms":
{
    System.out.println("Veuillez entrer les
numéros des symptômes (séparés par des virgules) ou '30' pour terminer
:");
    symptoms = Arrays.asList(
response.getContent().split(","));
    String Symptoms= getSymptoms();
    ACLMessage answer = new
ACLMessage(ACLMessage.INFORM);
    answer.addReceiver(response.getSender());
    answer.setConversationId("symptoms");
    answer.setContent(Symptoms);
    send(answer);

}
break;

case "prescription":
    System.out.println(response.getContent());
    System.out.println("je vais respecter la
prescription");
    // Step 6: Process the prescription
    // Implement your logic to process the
prescription here

    // Step 7: Send a goodbye message to the
secretary
    ACLMessage goodbye = new
ACLMessage(ACLMessage.INFORM);
    goodbye.setConversationId("goodbye-from-
patient");
    goodbye.addReceiver(response.getSender());
    goodbye.addReceiver(new AID("infirmière",
AID.ISLOCALNAME));
    goodbye.setContent("Aurevoir");
    send(goodbye);
    System.out.println("Agent patient dit aurevoir à
la secrétaire t au docteur");
    break;

case "goodbye-response":
    if(Objects.equals(response.getContent(),
"Doctor")){
        System.out.println("Merci docteur");
    }
    else if(Objects.equals(response.getContent(),

```

```
"Secretary")) {  
    System.out.println("Merci !");  
}  
break;  
  
  
case "notebook":  
    ACLMessage message = new  
ACLMessage(ACLMessage.INFORM);  
    message.addReceiver(response.getSender());  
    message.setConversationId("notebook-response");  
    String content=Notebook();  
    System.out.println("Voici mon carnet medical!" +  
content);  
    message.setContent(content);  
    send(message);  
break;  
  
default:  
    // Handle other cases here  
}  
}  
}  
}
```

Notebook

```
public class Notebook {  
  
    private String name;  
    private String size; //metre  
    private String gender;  
  
    public String getSize() {  
        return size;  
    }  
  
    public void setSize(String size) {  
        this.size = size;  
    }  
  
    public String getGender() {  
        return gender;  
    }  
  
    public void setGender(String gender) {  
        this.gender = gender;  
    }  
  
    public String getWeight() {  
        return weight;  
    }  
}
```

```

}

public void setWeight(String weight) {
    this.weight = weight;
}

public String getBloodType() {
    return bloodType;
}

public void setBloodType(String bloodType) {
    this.bloodType = bloodType;
}

public String getPatientName() {
    return patientName;
}

public void setPatientName(String patientName) {
    this.patientName = patientName;
}

public String getMedicalHistory() {
    return medicalHistory;
}

public void setMedicalHistory(String medicalHistory) {
    this.medicalHistory = medicalHistory;
}

public String getInsurance() {
    return insurance;
}

public void setInsurance(String insurance) {
    this.insurance = insurance;
}

public String getContactInfo() {
    return contactInfo;
}

public void setContactInfo(String contactInfo) {
    this.contactInfo = contactInfo;
}

public String getTemperature() {
    return temperature;
}

public void setTemperature(String temperature) {
    this.temperature = temperature;
}

public String getBloodPressure() {
    return bloodPressure;
}

public void setBloodPressure(String bloodPressure) {
    this.bloodPressure = bloodPressure;
}

```

```

public String getReason() {
    return reason;
}

public void setReason(String reason) {
    this.reason = reason;
}

private String weight; //kg
private String bloodType;
private String patientName;
private String medicalHistory;
private String insurance;
private String contactInfo;
private String temperature;
private String bloodPressure;
private String reason;

public Notebook(String name, String size, String gender, String
weight, String bloodType, String patientName, String medicalHistory,
String insurance, String contactInfo, String temperature, String
bloodPressure, String reason) {
    this.name=name;
    this.size = size;
    this.gender = gender;
    this.weight = weight;
    this.bloodType = bloodType;
    this.patientName = patientName;
    this.medicalHistory = medicalHistory;
    this.insurance = insurance;
    this.contactInfo = contactInfo;
    this.temperature = temperature;
    this.bloodPressure = bloodPressure;
    this.reason = reason;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String convertNotebookToJson() {

    String jsonBuilder = "{" +
        "\"name\":\"", + this.getName() + "\",\"" +
        "\"size\":\"", + this.getSize() + "\",\"" +
        "\"gender\":\"", + this.getGender() + "\",\"" +
        "\"weight\":\"", + this.getWeight() + "\",\"" +
        "\"bloodType\":\"", + this.getBloodType() + "\",\"" +
        "\"patientName\":\"", + this.getPatientName() + "\",\"" +
        "\"medicalHistory\":\"", + this.getMedicalHistory() +
        "\",\"" +
        "\"insurance\":\"", + this.getInsurance() + "\",\"" +
        "\"contactInfo\":\"", + this.getContactInfo() + "\",\"" +
        "\"temperature\":\"", + this.getTemperature() + "\",\"" +
        "\"bloodPressure\":\"", + this.getBloodPressure() + "\"," +
}

```

```
    "\"reason\":\"" + this.getReason() + "\",  
    \"}\";  
  
    return jsonBuilder;  
}  
  
}
```