

# Práctica 1. Primeros pasos con JADE

---

DISEÑO BASADO EN AGENTES CURSO 2023/2024 MANUEL JESÚS COBO MARTÍN



**UNIVERSIDAD  
DE GRANADA**



**MANUEL JESÚS COBO MARTÍN**

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL |  
UNIVERSIDAD DE GRANADA

# 1 Introducción a JADE

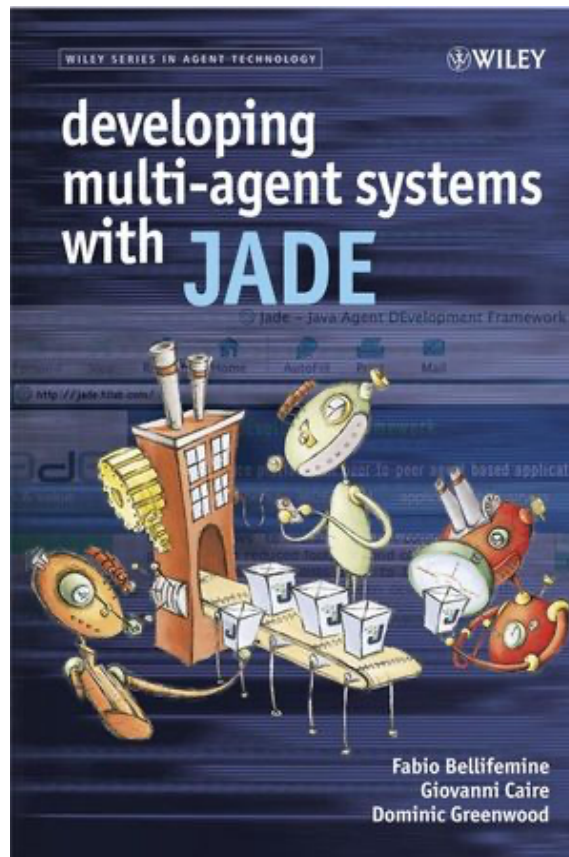
JADE (Java Agent DEvelopment Framework) es una plataforma de código libre p2p para el desarrollo de sistemas multi-agentes. JADE es un sistema middleware completamente distribuido, con una infraestructura flexible, lo que permite su fácil extensión a través de plugins. La plataforma nos permite el desarrollo de aplicaciones multi-agentes completas, ya que nos provee de un entorno de ejecución que controla todo el ciclo de vida de los agentes, la lógica interna de los agentes, así como un conjunto de herramientas gráficas tanto para la administración del servidor como para la gestión de los agentes.

Como su propio nombre indica, la plataforma JADE está desarrollada en Java, por lo que tendremos que emplear dicho lenguaje a la hora de desarrollar nuestro sistema multiagente.

La página principal de JADE es: <http://jade.tilab.com>, en ella podemos acceder a la documentación de la plataforma, así como ejemplos, e información extra. Por otro lado, los fundamentos de JADE tanto a nivel teóricos, como prácticos (programación), vienen explicados en detalle en el libro “developing multi-agent systems with JADE” publicado por Willey (<https://www.wiley.com/en-us/Developing+Multi+Agent+Systems+with+JADE-p-9780470058404>).



Figura 1. Página principal de JADE



*Figura 2. Libro sobre JADE*

## 2 JADE y el paradigma de los agentes

JADE es una plataforma software que proporciona funcionalidades básicas de capa intermedia que son independientes de la aplicación específica y que simplifican la realización de aplicaciones distribuidas que explotan la abstracción del agente de software. Un mérito significativo de JADE es que implementa esta abstracción sobre un lenguaje orientado a objetos muy conocido, Java, proporcionando una API sencilla y amigable. La abstracción del agente influye en las decisiones de diseño que se tomaron:

- Un agente es autónomo y proactivo: Un agente no puede proporcionar call-backs o su propio objeto de referencia a otros agentes para mitigar cualquier posibilidad de que otras entidades opten de forma simultánea el control de sus servicios. Un agente debe tener su propio hilo de ejecución, utilizándolo para controlar su ciclo de vida y decidir de forma autónoma cuándo realizar qué acciones.
- Los agentes pueden decir "no" y están desacoplados: La comunicación asíncrona basada en mensajes es la forma básica de comunicación entre agentes en JADE; un agente que desee comunicarse debe enviar un mensaje a un destino identificado (o conjunto de destinos). No existe dependencia temporal entre el emisor y los receptores: un receptor puede no estar disponible cuando el emisor emite el mensaje. Tampoco es necesario obtener la referencia de objeto de los agentes receptores, sino simplemente identidades de nombre que el sistema de transporte de mensajes es capaz de resolver en direcciones de transporte adecuadas. Incluso es posible que el emisor desconozca la identidad exacta de un receptor y que, en su lugar, defina un conjunto de receptores mediante una agrupación intencionada (por ejemplo, todos los agentes que prestan el servicio “profesor de DBA”) o mediada por un agente proxy (por ejemplo, propagar este mensaje a todos los agentes del dominio “dba.ugr.es”). Además, esta forma de comunicación permite al receptor seleccionar qué mensajes procesar y cuáles descartar, así como definir su propia

prioridad de procesamiento. También permite al emisor controlar su hilo de ejecución y así no quedar bloqueado hasta que el receptor procese el mensaje. Por último, también proporciona una ventaja interesante a la hora de implementar la comunicación multicast como una operación atómica en lugar de como N llamadas consecutivas a métodos (es decir, una operación de tipo send con una lista de múltiples receptores de mensajes en lugar de una llamada a un método por cada objeto remoto con el que se desea comunicar).

- El sistema es Peer-to-Peer: Cada agente se identifica mediante un nombre único global (el AgentIdentifier, o AID, según la definición de FIPA). Puede entrar y salir de una plataforma anfitriona en cualquier momento y puede descubrir otros agentes a través de servicios de página blanca y amarilla (proporcionados en JADE por AMS y los agentes DF, tal y como se definen también en las especificaciones FIPA). Un agente puede iniciar una comunicación con cualquier otro agente en cualquier momento que lo desee y puede igualmente ser objeto de una comunicación entrante en cualquier momento.

Basándose en estas decisiones de diseño, JADE se implementó para proporcionar a los programadores las siguientes funcionalidades básicas, listas para usar y fáciles de personalizar.

- Un sistema totalmente distribuido habitado por agentes, cada uno de los cuales se ejecuta como un hilo independiente, potencialmente en diferentes máquinas remotas, y capaz de comunicarse entre sí de forma transparente, es decir, la plataforma proporciona una API única independiente de la ubicación que abstrae la infraestructura de comunicación subyacente.
- Plena conformidad con las especificaciones de la FIPA. La plataforma participó con éxito en todos los eventos de interoperabilidad de la
- Transporte eficiente de mensajes asíncronos a través de una API transparente a la localización. La plataforma selecciona el mejor medio de comunicación disponible y, en la medida de lo posible, evita marshalling/unmarshalling de objetos java. Al cruzar los límites de la plataforma, los mensajes se transforman automáticamente desde la propia representación Java interna de JADE a sintaxis, codificaciones y protocolos de transporte adecuados y conformes con la FIPA.
- Implementación de páginas blancas y amarillas. Los sistemas federados pueden implementarse para representar dominios y subdominios como un grafo de directorios federados.
- Gestión sencilla y eficaz del ciclo de vida de los agentes. Cuando se crean los agentes, se les asigna automáticamente un identificador único global y una dirección de transporte que se utilizan para registrarse en el servicio de páginas blancas de su plataforma. También se proporcionan API sencillas y herramientas gráficas para gestionar local y remotamente los ciclos de vida de los agentes, es decir, crear, suspender, reanudar, congelar, descongelar, migrar, clonar y matar.
- Movilidad de los agentes. Tanto el código del agente como, bajo ciertas restricciones, el estado del agente, pueden migrar entre procesos y máquinas. La migración de agentes es transparente para los agentes en comunicación, que pueden seguir interactuando incluso durante el proceso de migración.
- Un mecanismo de suscripción para agentes, e incluso aplicaciones externas, que deseen suscribirse a una plataforma para ser notificados de todos los eventos de la plataforma, incluyendo eventos relacionados con el ciclo de vida y eventos de intercambio de mensajes.
- Un conjunto de herramientas gráficas para ayudar a los programadores a depurar y supervisar.
- Soporte de ontologías y lenguajes de contenido. La plataforma realiza automáticamente la comprobación de ontologías y la codificación de contenidos, y los programadores pueden seleccionar los lenguajes de contenidos y ontologías que prefieran (por ejemplo, basados en XML y RDF). Los programadores también pueden implementar nuevos lenguajes de contenido para satisfacer requisitos específicos de la aplicación.
- Biblioteca de protocolos de interacción que modelan patrones típicos de comunicación orientados a la consecución de uno o varios objetivos. Los esqueletos independientes de

la aplicación están disponibles como un conjunto de clases Java que pueden personalizarse con código específico de la aplicación. Los protocolos de interacción también pueden representarse e implementarse como un conjunto de máquinas de estados finitos concurrentes.

- Integración con diversas tecnologías basadas en web, como JSP, servlets, applets y tecnología de servicios web.
- La plataforma también puede configurarse fácilmente para atravesar cortafuegos y utilizar sistemas NAT.
- Compatibilidad con la plataforma J2ME y el entorno inalámbrico. JADE se puede ejecutar en las plataformas J2ME-CDC y-CLDC a través de un conjunto uniforme de API que abarca tanto los entornos J2ME como J2SE.
- Una interfaz dentro del proceso para lanzar/controlar una plataforma y sus componentes distribuidos desde una aplicación externa.
- Un núcleo extensible diseñado para permitir a los programadores ampliar la funcionalidad de la plataforma mediante la adición de servicios distribuidos a nivel de núcleo. Este mecanismo se inspira en el enfoque de la programación orientada a aspectos, que permite entretejer distintos aspectos en el código de la aplicación y coordinarlos a nivel del núcleo.

### 3 Arquitectura de JADE

La plataforma JADE se compone de contenedores de agentes que pueden distribuirse por la red. Los agentes viven en contenedores que son el proceso Java que proporciona el entorno de ejecución de JADE y todos los servicios necesarios para alojar y ejecutar agentes. Existe un contenedor especial, denominado *main container*, que representa el punto de arranque de la plataforma: es el primer contenedor que se lanza y todos los demás contenedores deben unirse al contenedor principal registrándose en él.

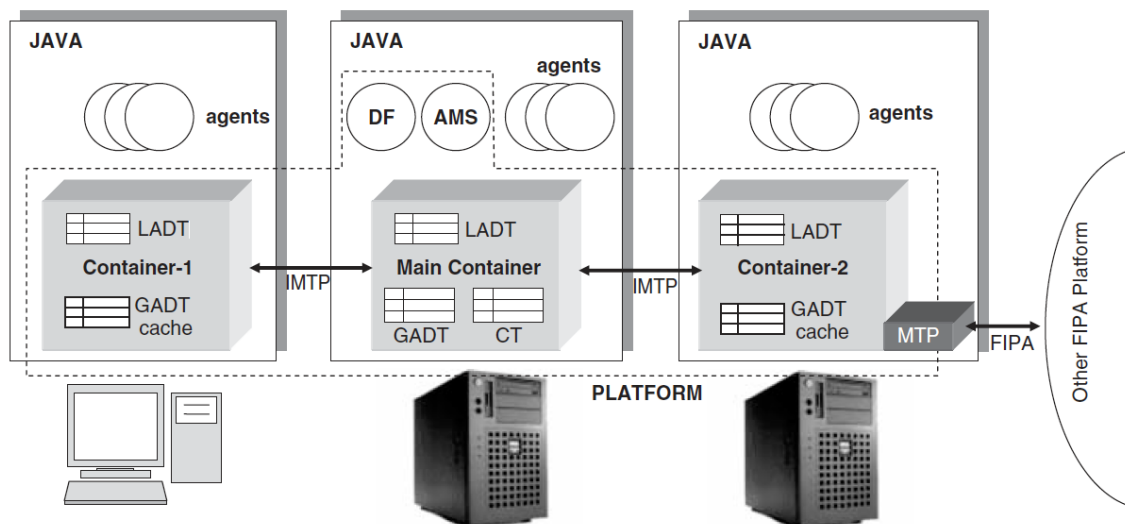


Figura 3. Relación entre los elementos principales de la plataforma JADE

El programador identifica los contenedores simplemente utilizando un nombre; por defecto, el contenedor principal se denomina "*Main container*", mientras que los demás se denominan "Contenedor-1", "Contenedor-2", etc. Existen opciones de línea de comandos para modificar los nombres por defecto. Como punto de arranque, el contenedor principal tiene las siguientes responsabilidades especiales:

- Gestionar la tabla de contenedores (CT), que es el registro de las referencias a objetos y direcciones de transporte de todos los nodos contenedores que componen la plataforma

- Gestionar la tabla descriptiva global de agentes (GADT), que es el registro de todos los agentes presentes en la plataforma, incluido su estado y ubicación actuales;
- Alojar el AMS y el DF, los dos agentes especiales que proporcionan el servicio de gestión de agentes y página blanca, y el servicio de página amarilla por defecto de la plataforma, respectivamente.

La identidad del agente está contenida en un Identificador de Agente (AID), que cumple con la normativa FIPA. Los elementos más básicos del AID son el nombre del agente y sus direcciones. El nombre de un agente es un identificador único global que JADE construye concatenando un apodo definido por el usuario (también conocido como nombre local, ya que es suficiente para desambiguar la comunicación intraplataforma) con el nombre de la plataforma. Las direcciones de los agentes son direcciones de transporte heredadas por la plataforma, donde cada dirección de plataforma corresponde a un punto final MTP (Protocolo de Transporte de Mensajes) donde se pueden enviar y recibir mensajes compatibles con FIPA. También se permite a los programadores añadir sus propias direcciones de transporte al AID cuando, para cualquier propósito específico de la aplicación, deseen implementar su propio MTP privado.

Cuando se lanza el contenedor principal, JADE instala e inicia automáticamente dos agentes especiales, cuyas funciones están definidas por el estándar FIPA de Gestión de Agentes:

1. El *Agent Managent Systems* (AMS) es el agente que supervisa toda la plataforma. Es el punto de contacto para todos los agentes que necesitan interactuar para acceder a las páginas blancas de la plataforma, así como para gestionar su ciclo de vida. Todos los agentes deben registrarse en el AMS (lo que JADE realiza automáticamente al poner en marcha el agente) para obtener un AID válido.
2. El *Directory Facilitator* (DF) es el agente que implementa el servicio de páginas amarillas, utilizado por cualquier agente que desee registrar sus servicios o buscar otros servicios disponibles. El DF de JADE también acepta suscripciones de agentes que deseen ser notificados cada vez que se produzca un registro o modificación de un servicio que coincida con algún criterio especificado. Se pueden iniciar varios DF simultáneamente para distribuir el servicio de páginas amarillas en varios dominios. Estos DF pueden federarse, si es necesario, estableciendo registros cruzados entre ellos que permitan la propagación de las solicitudes de los agentes a través de toda la federación.

## 4 Configuración del entorno

Para la realización de las prácticas de la asignatura necesitaremos adecuar nuestro entorno de desarrollo, para poder programar, compilar y ejecutar nuestro sistema multi-agente mediante la plataforma JADE. Para ello, necesitaremos:

- Java. Bien el jdk de Oracle o openJDK.
- Un entorno de desarrollo como Netbeans, IntelliJ o Eclipse. La recomendación es usar Netbeans, ya que es de código libre, no consume muchos recursos, y es fácil de utilizar. Pero perfectamente se pueden emplear, siempre que se sepan usar de un modo adecuado, IDEs más potentes como IntelliJ.
- JADE. En la web de JADE, en la sección de descargas podremos encontrar un comprimido con todo. Tendremos que descomprimir el fichero y después los binarios y la documentación. Es recomendable guardarlo todo en una misma carpeta. Dicha carpeta la utilizaremos como librería base para todos los proyectos que vayamos a construir.

Una vez que tengamos todo el software en nuestro ordenador, tendremos que o bien modificar la variable de entorno CLASSPATH para indicar donde tenemos JADE instalado, o configurar el IDE para que pueda usar JADE.

Lo más sencillo quizás es la segunda opción. Así que en Netbeans vamos a crear una nueva librería (Tools->Libraries, y después New Library) (que llamaremos JADE, JADElib o cualquier otro nombre significativo), y añadir tanto los .jar como la documentación.

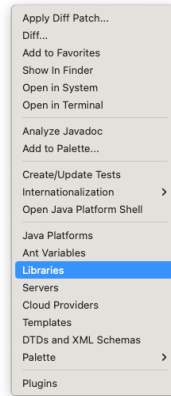


Figura 4. Añadir librería en Netbeans

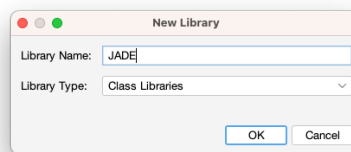


Figura 5. Crear nueva librería

## 5 Ejecución del servidor JADE

Para ejecutar el contenedor principal debemos de ejecutar el siguiente comando:

```
java -cp jade.jar jade.Boot -name dba_server -gui
```

- Usaremos -cp sino tenemos incluidos los binarios de jade en el CLASSPATH.
- Mediante -gui activamos la interfaz gráfica del servido
- Mediante -name indicamos el nombre del servidor

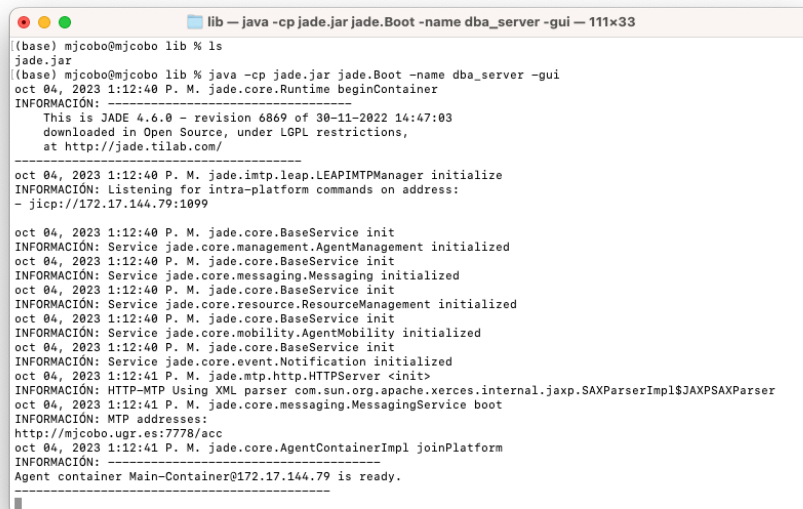


Figura 6. Salida del comienzo de la ejecución del sistema



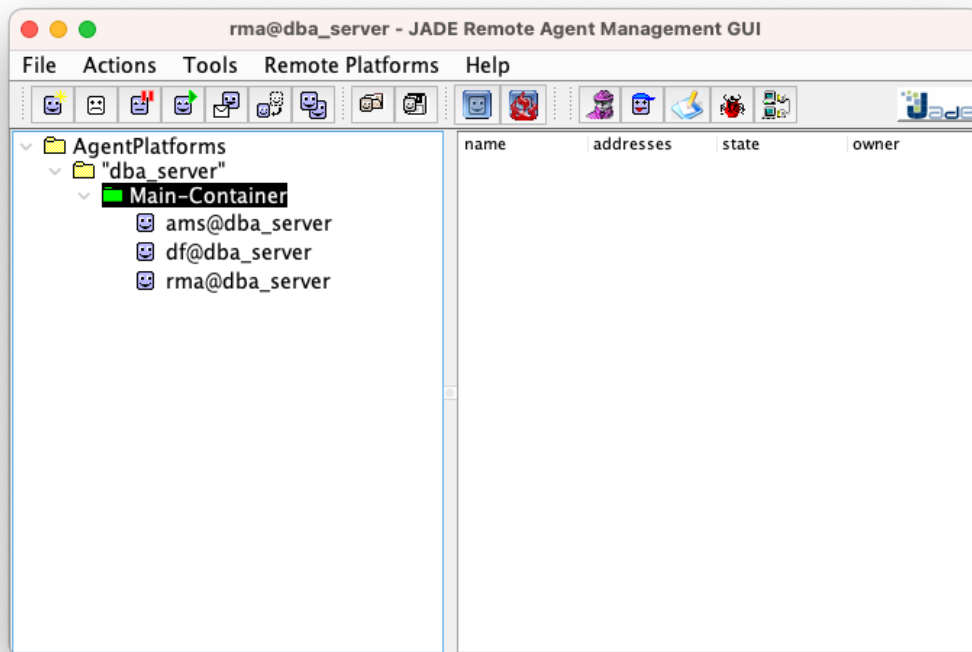


Figura 7. Ventana principal del JADE Remote Agent Management GUI

## 6 Primer agente

Como ejemplo, vamos a crear un primer agente que simplemente diga “hola”. Para ello, tendremos que crearnos un proyecto en Netbeans (un proyecto Ant clásico), y añadir la librería JADE que previamente habremos configurado.

De momento vamos a crear una única clase que se llame HelloWorldAgent (o cualquier otro nombre significativo). La clase extenderá la clase Agent de la librería JADE.

```
import jade.core.Agent;
```

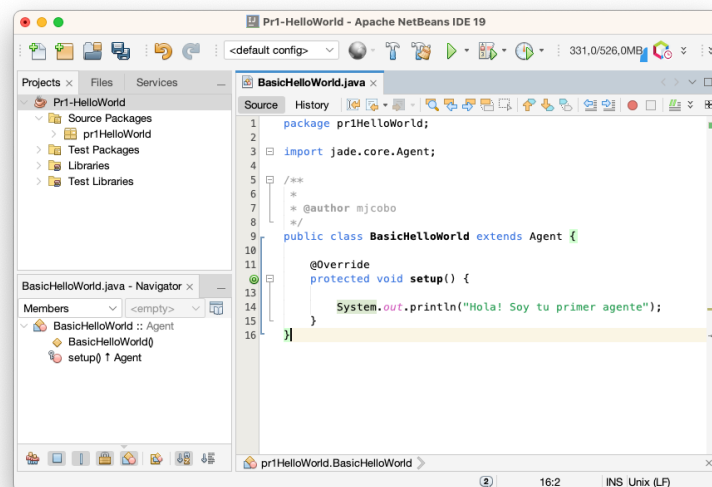


Figura 8. Agente básico

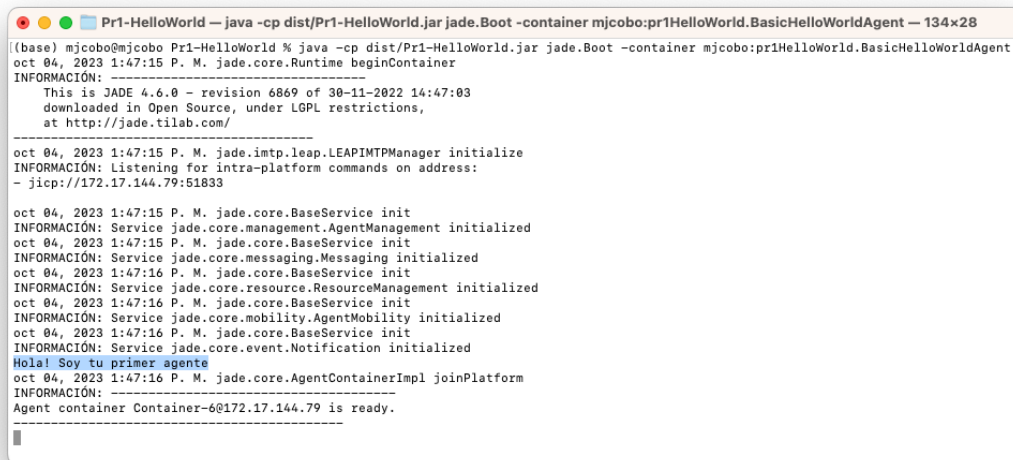


Para ejecutar nuestro agente, tendremos que lanzarlo desde la terminal mediante el siguiente comando:

```
java -cp dist/Pr1-HelloWorld.jar jade.Boot -container  
mjcobo:pr1HelloWorld.BasicHelloWorldAgent
```

- -container nos indica que no vamos a lanzar un contenedor principal. Si ya tuviéramos el main ejecutándose, y intentásemos lanzar nuestro agente sin este argumento, obtendríamos un error.
- La parte final es nombreAgente:clase. El nombre del agente es el nombre que le vamos a dar en el sistema, y la clase es la ruta absoluta hasta esa clase.
- Si tenemos varios agentes podríamos ejecutarlos añadiendo -agents y separando cada agentes por ;.

Ejecutando el comando anterior, obtendríamos la siguiente salida:



```
Pr1-HelloWorld — java -cp dist/Pr1-HelloWorld.jar jade.Boot -container mjcobo:pr1HelloWorld.BasicHelloWorldAgent — 134x28
(base) mjcobo@mjcobo Pr1-HelloWorld % java -cp dist/Pr1-HelloWorld.jar jade.Boot -container mjcobo:pr1HelloWorld.BasicHelloWorldAgent
oct 04, 2023 1:47:15 P. M. jade.core.Runtime beginContainer
INFORMACIÓN: -----
This is JADE 4.6.0 - revision 6869 of 30-11-2022 14:47:03
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----
oct 04, 2023 1:47:15 P. M. jade.imtp.leap.LEAPIMTPManager initialize
INFORMACIÓN: Listening for intra-platform commands on address:
- jicp://172.17.144.79:51833
oct 04, 2023 1:47:15 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.management.AgentManagement initialized
oct 04, 2023 1:47:15 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.messaging.Messaging initialized
oct 04, 2023 1:47:16 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.resource.ResourceManagement initialized
oct 04, 2023 1:47:16 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.mobility.AgentMobility initialized
oct 04, 2023 1:47:16 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.event.Notification initialized
Hola! Soy tu primer agente
oct 04, 2023 1:47:16 P. M. jade.core.AgentContainerImpl joinPlatform
INFORMACIÓN: -----
Agent container Container-6@172.17.144.79 is ready.
```

Figura 9. Ejecución del agente básico

En la interfaz gráfica del servidor podremos ver que se ha creado un contenedor distinto al principal, que contiene nuestro agente:

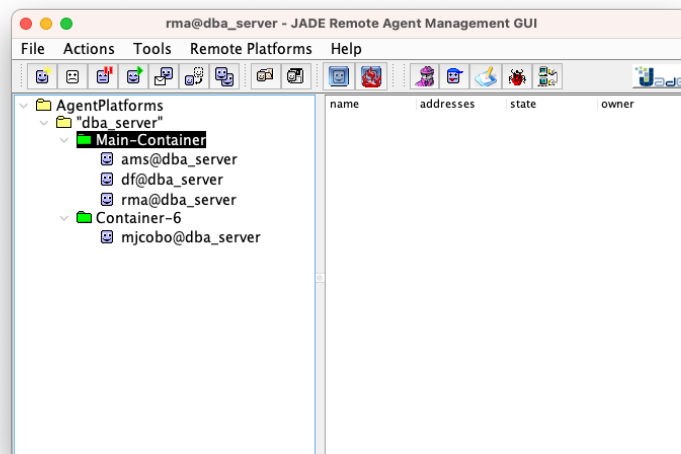


Figura 10. GUI con el agente hello

Como podemos observar, el agente realiza su tarea, pero el proceso nunca termina. Para que termine, tendremos que forzarlo. Esto es debido a que hemos creado un contenedor de agentes ad-hoc, y estará activo mientras que no forcemos su cierre. Esto lo podremos hacer mediante `ctrl-c` en la terminal, o matando (kill) el contenedor del agente en la interfaz gráfica.

## 7 Ciclo de vida de un agente

Los agentes tienen un ciclo de ejecución específico, teniendo métodos que se llaman al inicio, y métodos que se llaman justo antes de terminar:

- El método `setup()` se encarga de la inicialización del agente.
- Para que un agente termine, se tiene que llamar al método `doDelete()`. Si durante el proceso, nuestro agente ha reservado recursos que tiene que liberar, se puede sobrescribir el método `takeDown()` que se llamará justo antes de que el agente termine.

Vamos a modificar nuestro agente básico para mostrar más información y ver cómo podemos programar la transición entre estados.

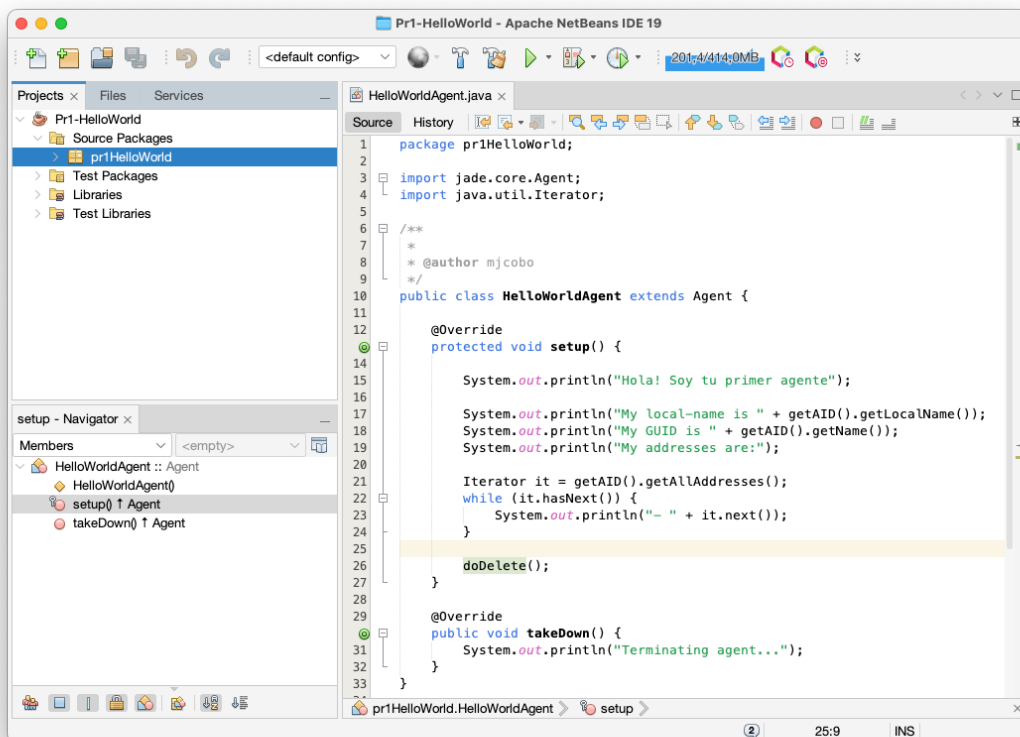


Figura 11. Agente básico extendido con `doDelete()` y `takeDown()`

En este caso, el agente durante el `setup()` imprime un mensaje de bienvenida, y muestra su nombre local, su GUID, y las direcciones que tiene asociadas. Finalmente llama al método `doDelete()` que al terminar invoca al método `takeDown()`, el cual, simplemente muestra un mensaje de despedida.

```

Pr1-HelloWorld — java -cp dist/Pr1-HelloWorld.jar jade.Boot -container mjcobo:pr1HelloWorld.HelloWorldAgent — 134x32
(base) mjcobo@mjcobo Pr1-HelloWorld % java -cp dist/Pr1-HelloWorld.jar jade.Boot -container mjcobo:pr1HelloWorld.HelloWorldAgent
oct 04, 2023 4:51:35 P. M. jade.core.Runtime beginContainer
INFORMACIÓN: -----
This is JADE 4.6.0 - revision 6869 of 30-11-2022 14:47:03
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----
oct 04, 2023 4:51:35 P. M. jade.imtp.leap.LEAPIMTPManager initialize
INFORMACIÓN: Listening for intra-platform commands on address:
- jicp://172.17.144.79:51514

oct 04, 2023 4:51:35 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.management.AgentManagement initialized
oct 04, 2023 4:51:35 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.messaging.Messaging initialized
oct 04, 2023 4:51:35 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.resource.ResourceManagement initialized
oct 04, 2023 4:51:35 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.mobility.AgentMobility initialized
oct 04, 2023 4:51:35 P. M. jade.core.BaseService init
INFORMACIÓN: Service jade.core.event.Notification initialized
Hola! Soy tu primer agente
oct 04, 2023 4:51:36 P. M. jade.core.AgentContainerImpl joinPlatform
INFORMACIÓN: -----
Agent container Container-1@172.17.144.79 is ready.
-----
My local-name is mjcobo
My GUID is mjcobo@dba_server
My addresses are:
- http://mjcobo.ugr.es:7778/acc
Terminating agent...

```

Figura 12. Ejecución del agente básico extendido con *doDelete()* y *takeDown()*

Como hemos de esperar, lo normal no es que el agente realice su tarea dentro del método *setup*. Por el contrario, la actividad se debería de hacer al finalizar la configuración inicial, y antes de terminar. Así, el flujo de trabajo de un agente será el siguiente:

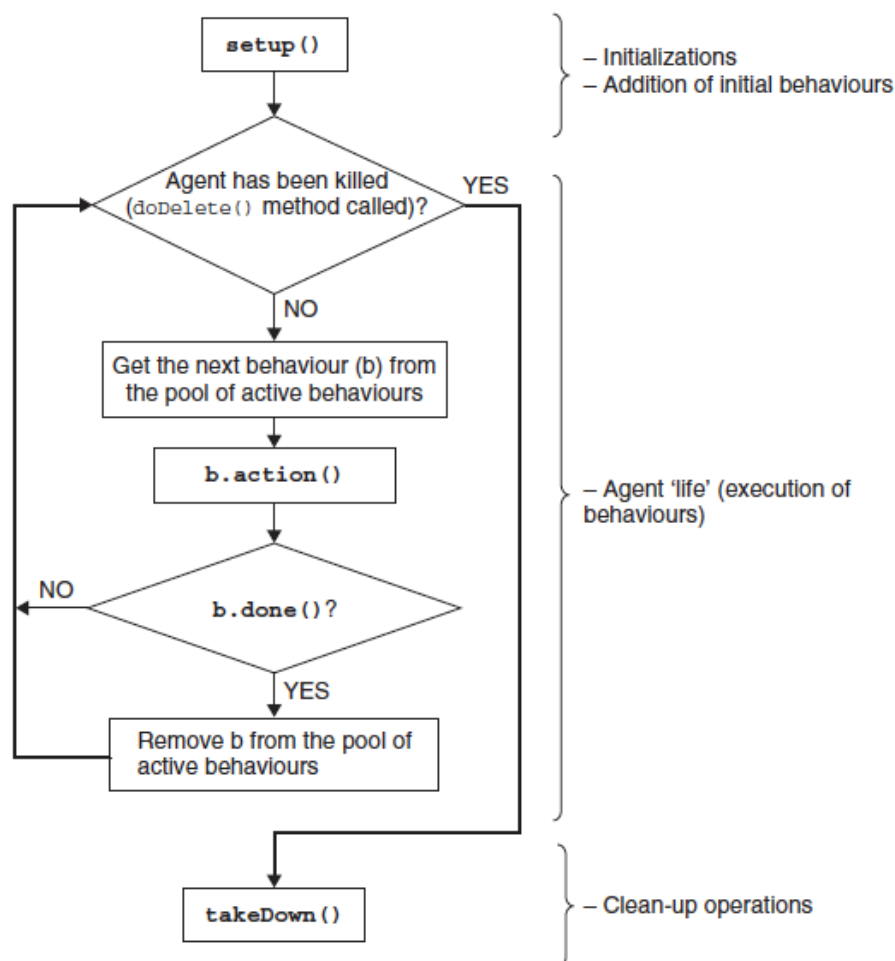


Figura 13. Ciclo de ejecución de un agente

La tarea específica del agente se realiza mediante los llamados **Behaviours**. Un comportamiento representa una tarea que un agente puede llevar a cabo y se implementa como un objeto de una clase que extiende `jade.core.behaviours.Behaviour`. Para que un agente ejecute la tarea implementada por un objeto `behaviour`, éste debe ser añadido al agente mediante el método `addBehaviour()` de la clase `Agent`. Los comportamientos pueden ser añadidos en cualquier momento cuando un agente arranca (en el método `setup()`) o desde dentro de otros comportamientos.

Cada clase que extienda `Behaviour` debe implementar dos métodos abstractos. El método `action()` define las operaciones a realizar cuando el comportamiento está en ejecución. El método `done()` devuelve un valor booleano que indica si un comportamiento ha finalizado y debe ser eliminado del conjunto de comportamientos que está ejecutando un agente.

## 7.1 Tipos de comportamientos

Principalmente, JADE contiene tres tipos de comportamientos:

- **One-shot.** Se ejecutan una única vez. La clase `jade.core.behaviours.OneShotBehaviour` ya implementa el método `done()`, el cual devuelve siempre `true`.
- **Cyclic.** Al contrario que el anterior comportamiento, este es un comportamiento cíclico que nunca termina, ya que la clase `jade.core.behaviours.CyclicBehaviour` implementa el método `done()` para que siempre devuelva `false`.
- **Comportamientos genéricos.** Es el programador el encargado de determinar cuándo tiene que terminar el comportamiento. Por ejemplo, cuando se hayan ejecutado `n` ciclos, o cuando se haya alcanzado cierto estado.
- El `WakerBehaviour` tiene métodos `action()` y `done()` preimplementados para ejecutar el método abstracto `onWake()` después de que expire un tiempo de espera determinado (especificado en el constructor). Tras la ejecución del método `onWake()`, el comportamiento se completa.
- El `TickerBehaviour` tiene los métodos `action()` y `done()` preimplementados para ejecutar el método abstracto `onTick()` repetidamente, esperando un periodo determinado (especificado en el constructor) después de cada ejecución. Un `TickerBehaviour` nunca se completa a menos que se elimine explícitamente o se llame a su método `stop()`.

## 8 Ejecución de un agente desde código

Anteriormente hemos visto como ejecutar un agente JADE desde la línea de comandos. Pero, aunque esta opción tiene mucha versatilidad, no siempre es la más útil o cómoda. Así, JADE dispone de ciertas clases y métodos que nos van a permitir lanzar o ejecutar los agentes, directamente desde código.

En primer lugar, tenemos que obtener la instancia del entorno de ejecución (`jade.core.Runtime`).

```
Runtime rt = Runtime.instance();
```

Después, deberemos crear un contenedor de agentes, del mismo modo que hacíamos con la línea de comandos. Para ello tendremos que indicarle, si queremos o son necesarios, algunos parámetros. Para ello, emplearemos un `Profile` (`jade.core.Profile`), el cual es una estructura que asocia un nombre a un valor. Los nombres podremos obtenerlos en las constantes proporcionadas en la clase `Profile` del core de JADE.

```
Profile p = new ProfileImpl();

p.setParameter(Profile.MAIN_HOST, host);
p.setParameter(Profile.CONTAINER_NAME, containerName);

ContainerController cc = rt.createAgentContainer(p);
```

Una vez tenemos el controlador del contenedor creado, simplemente debemos crear un nuevo agente, y empezar su ejecución.

```
AgentController ac = cc.createNewAgent(agentName,
    HelloWorldAgent.class.getCanonicalName(), null);

ac.start();
```

## 9 Ejercicios a resolver

Para la resolución de la práctica, deberemos de resolver los siguientes ejercicios:

1. Crear un Agente básico que muestre un mensaje por consola. (1pto)
2. Crear un Agente usando comportamientos que muestre un mensaje por consola una única vez. (2.5ptos)
3. Crear un Agente básico que muestre un mensaje de forma indefinida, pero esperando 2 segundos entre mensaje y mensaje. (2.5ptos)
4. Crear un Agente que solicite el número de elementos numéricos a leer, los sume, y calcule su media. Se deberá implementar utilizando una secuencia de comportamientos. (4ptos)

La práctica se evaluará en clase de prácticas, comprobando el correcto funcionamiento y ejecución tanto del entorno como de los agentes. El código se tendrá que entregar mediante la tarea habilitada en PRADO.

La entrega podrá realizarse hasta el miércoles 18 de octubre.