Práctica 1: Introducción a Jade
Curso 2015/16

Arquitectura. Herramientas. Agentes sencillos.

Jade

- Jade (Java agents development environment)
  - Jade es un middleware (TILAB) para el desarrollo de aplicaciones distribuidas multiagente.
  - Proporciona tanto el entorno de desarrollo para la creación de aplicaciones basadas en agentes como el entorno de ejecución.
  - Jade presenta las siguientes características:
    - Arquitectura P2P.
    - Interoperabilidad: Jade cumple con las especificaciones FIPA
    - Portabilidad: La API es independiente de la red sobre la que se opera y de la versión de java utilizada (J2EE, J2SE, J2ME)
    - Intuitiva: Jade se ha desarrollado para ofrecer una API fácil de aprender y sencilla de manejar.

Jade versión actual

- La versión actual es la 4.3.3 (Diciembre 2014)
- http://jade.tilab.com
- La distribución de Jade se compone de varios ficheros zip:
  - JADE-doc-4.3.3.zip → documentación bastante completa.
  - JADE-src-4.3.3.zip → códigos fuente java de JADE.
  - JADE-examples-4.3.3.zip → carpeta con ejemplos de aplicaciones en Jade.
  - JADE-bin-4.3.3.zip → las APIs básicas para poder programar y ejecutar agentes y en entorno Jade.
  - JADE-all-4.3.3.zip → todos los ficheros anteriores.

Jade arquitectura

- Los agentes JADE necesitan del entorno de ejecución donde poder estar pertenecer.
- Cada instancia del entorno de ejecución se denomina *contenedor* (**container**) y al conjunto de los contenedores se le denomina *plataforma* (**plattform**).
- En cada plataforma debe existir un contenedor especial denominado *contenedor principal* (**main container**), La principal diferencia del contenedor principal respecto al resto de contenedores es que alberga dos agentes especiales:
  - **AMS** (Agent Management System): Este agente proporciona el servicio de nombres, asegurando que cada agente en la plataforma disponga de un nombre único.
  - **DF** (Directory facilitator): Proporciona el servicio de páginas amarillas (organiza los agentes que los deseen por los servicios que proporcionan al resto de agentes).

Arquitectura Jade

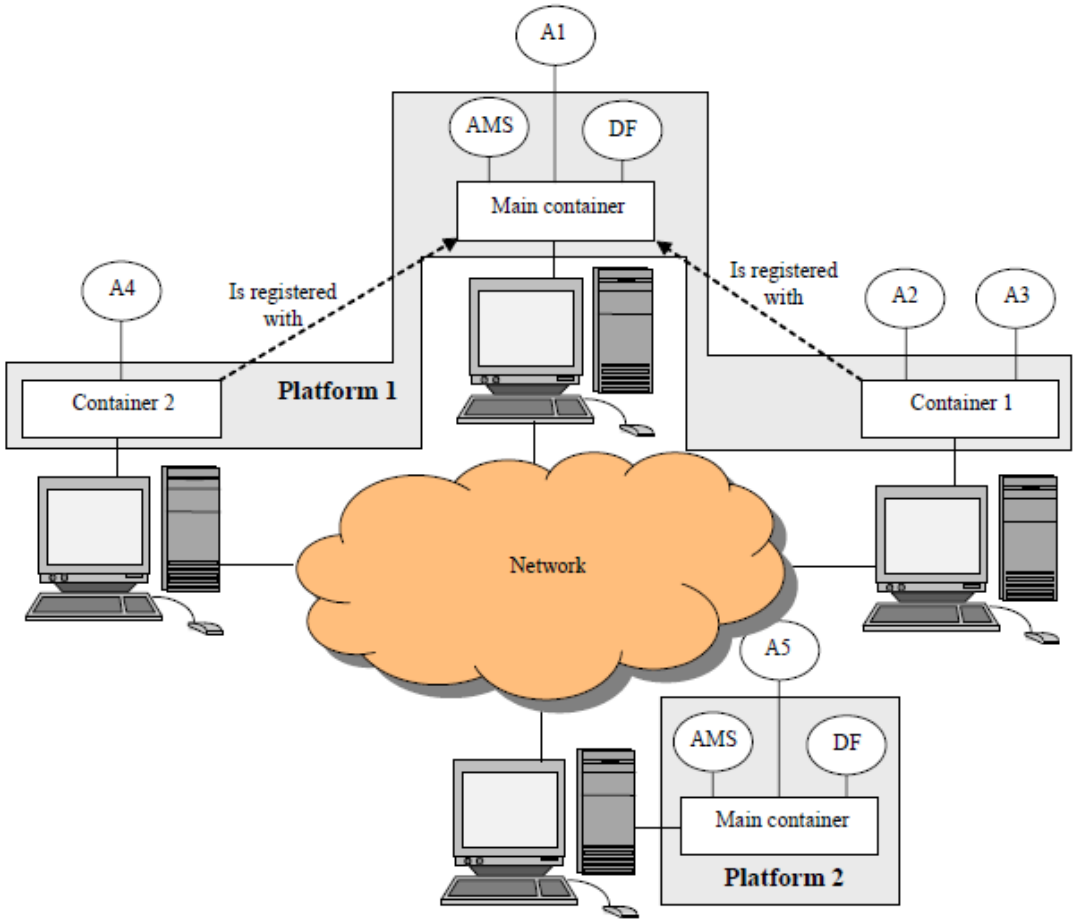
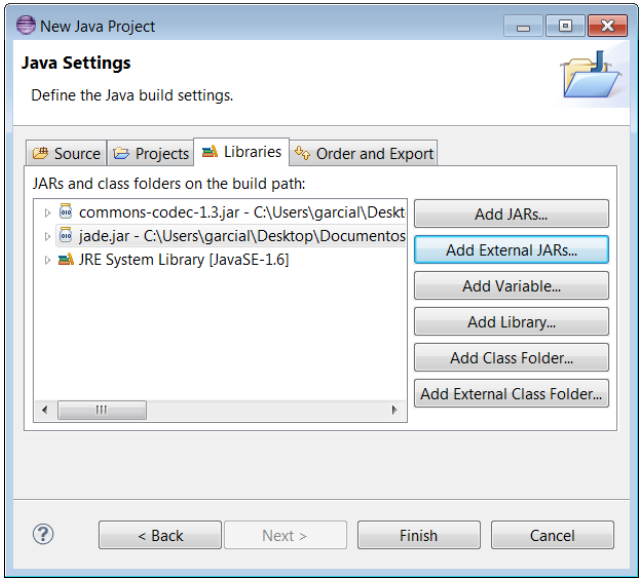
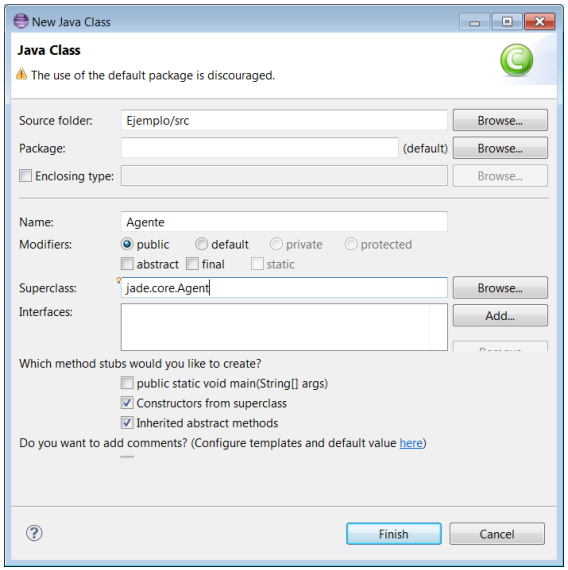


Figure 1Containers and Platforms

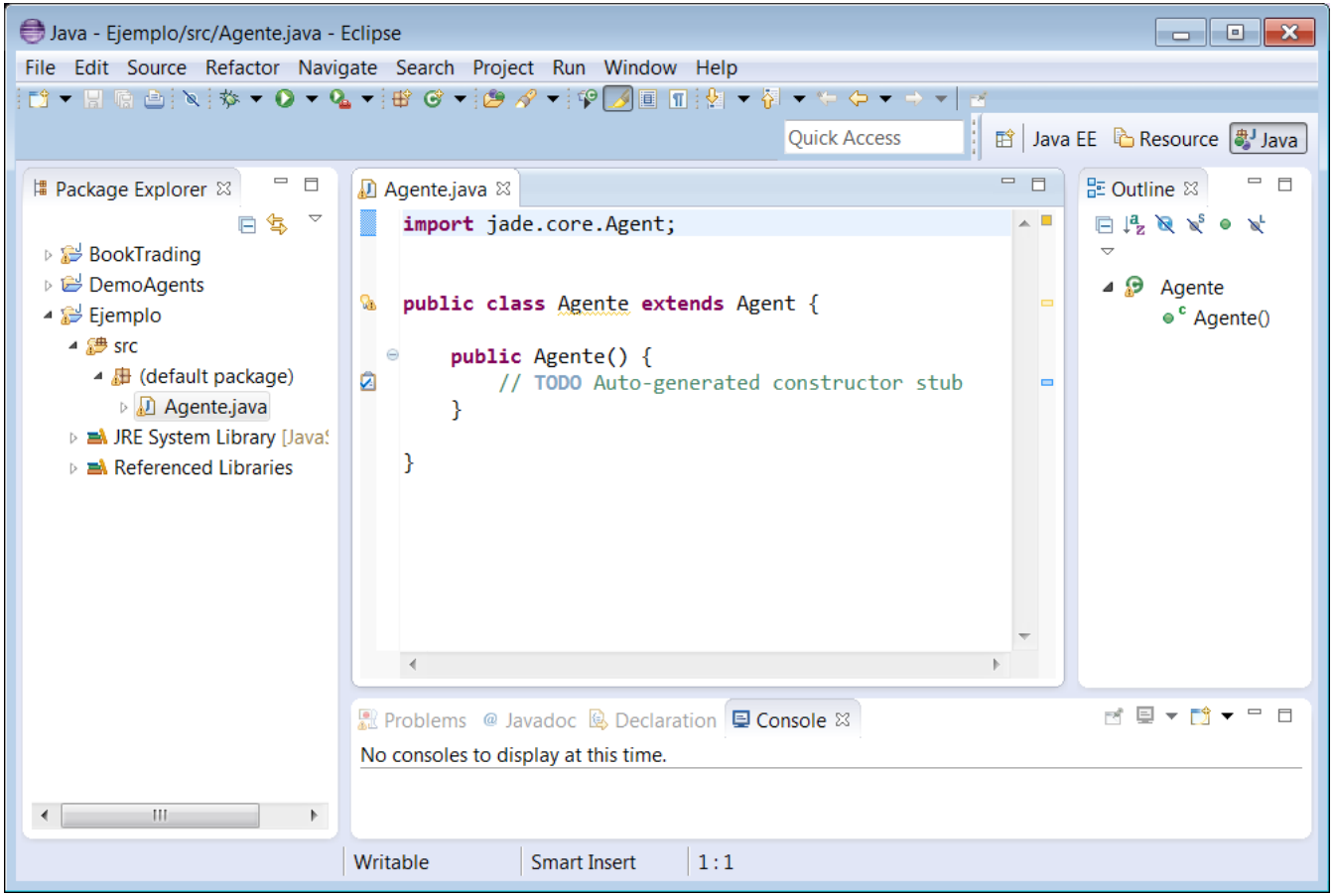
Programando Agentes

- Descomprime el archivo *JADE-bin-4.3.3.zip* en un directorio.
- Añade al CLASSPATH los ficheros **.jar* que hay en el directorio /lib donde has descomprimido *JADE*.
- En el IDE Eclipse hay que crear un nuevo proyecto al que hay que añadir las jars externas de JADE.
- Una vez creado un proyecto puedes crear ficheros java en cada uno de los cuales programarás agentes.





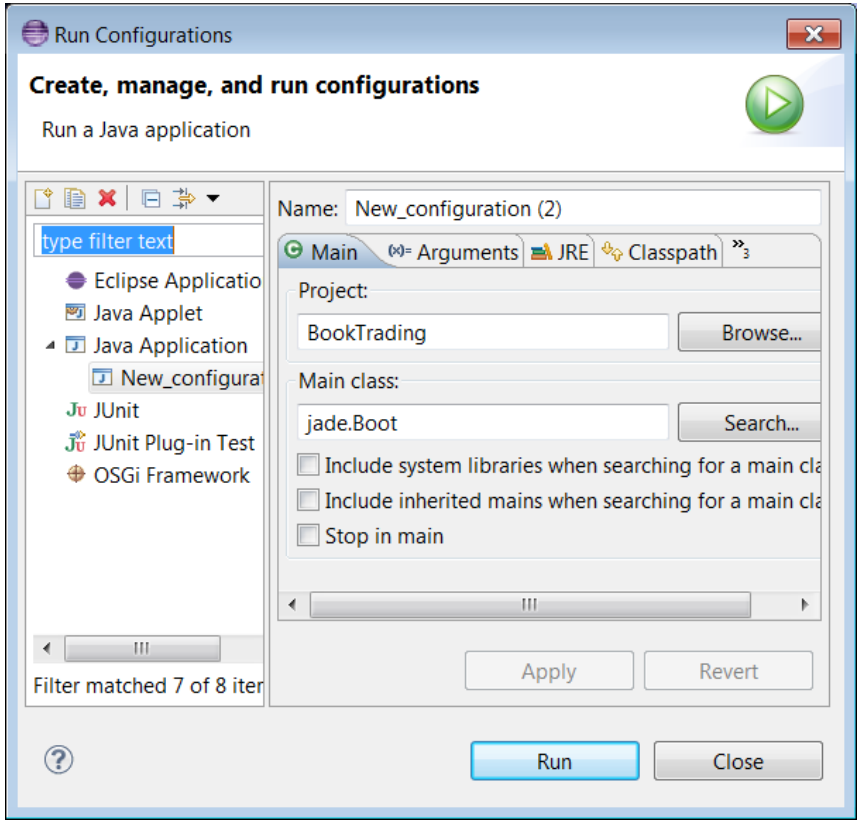
Programando Agentes

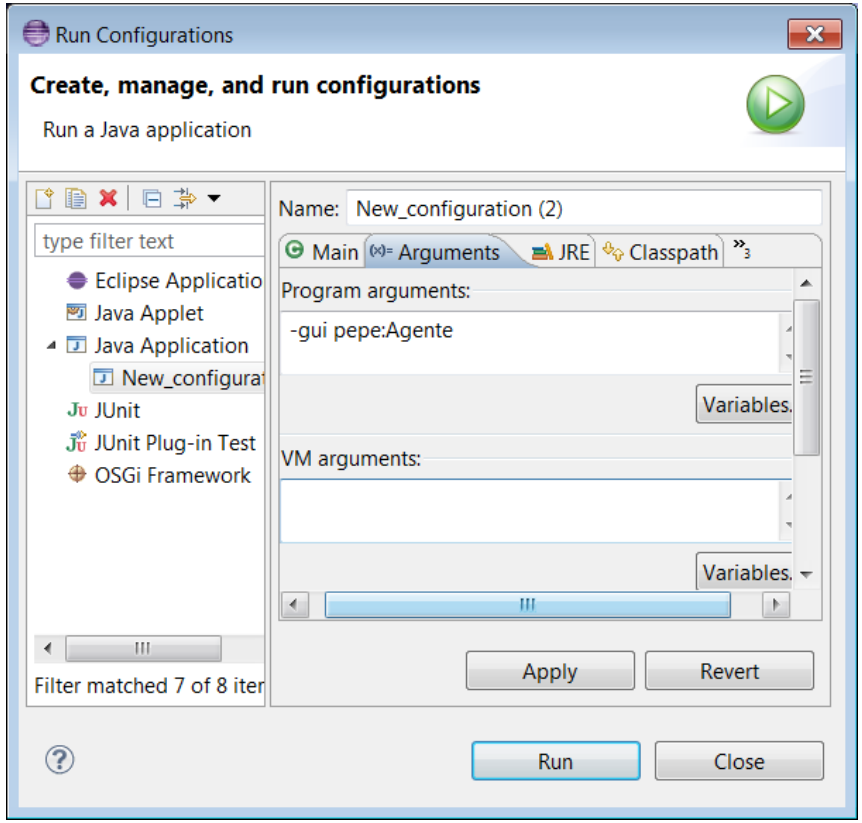


Programando Agentes

- La clase jade.core.Agent
  - Todos los agentes extienden la clase Agent.
  - Cada agente tiene un identificador único en la plataforma (nombre@host:nnnn/JADE).
  - Al crearse el agente se realizan varias tareas automáticamente:
    - Se llama al constructor
    - Se crea un identificador único (AID)
    - Se registra el agente en el AMS
    - Se ejecuta el método setup() que únicamente debe contener el código relativo a tareas de inicialización.

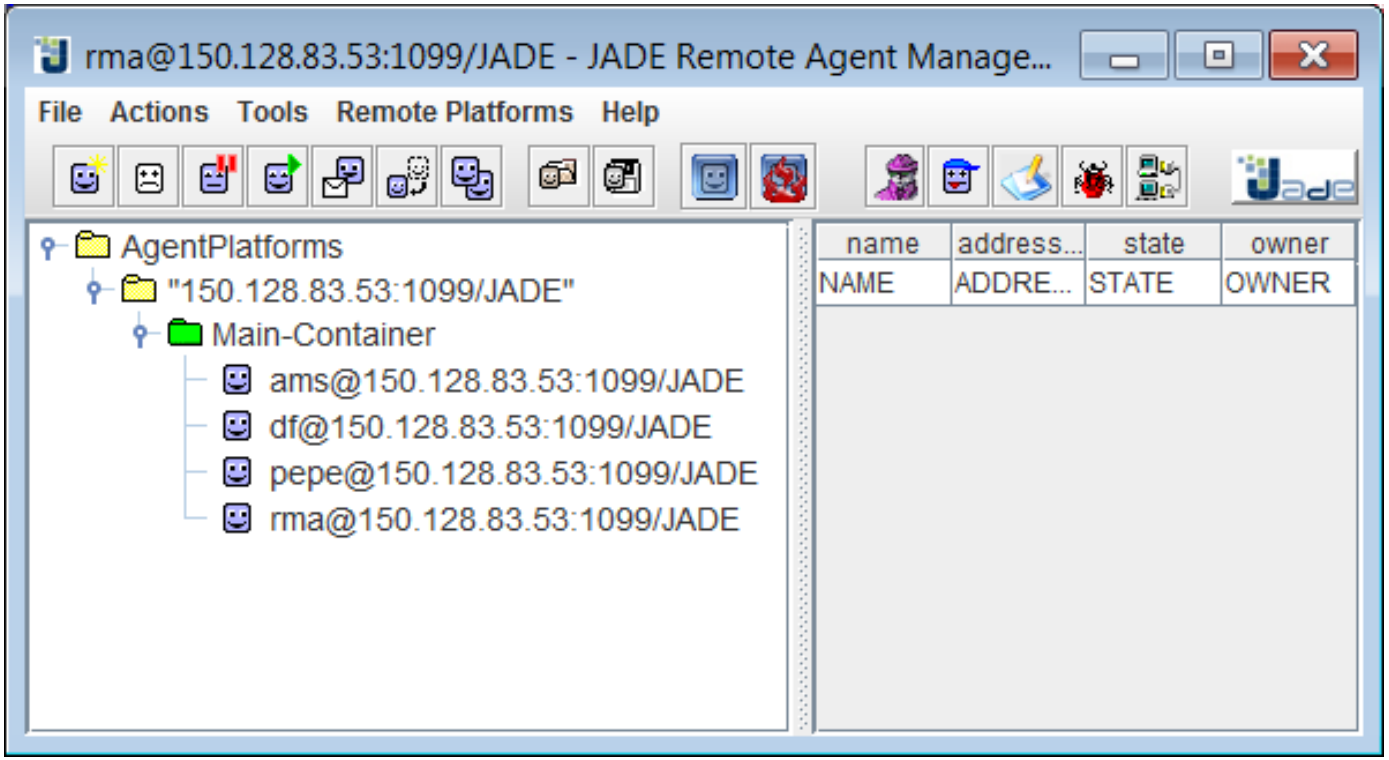
Ejecutando agentes





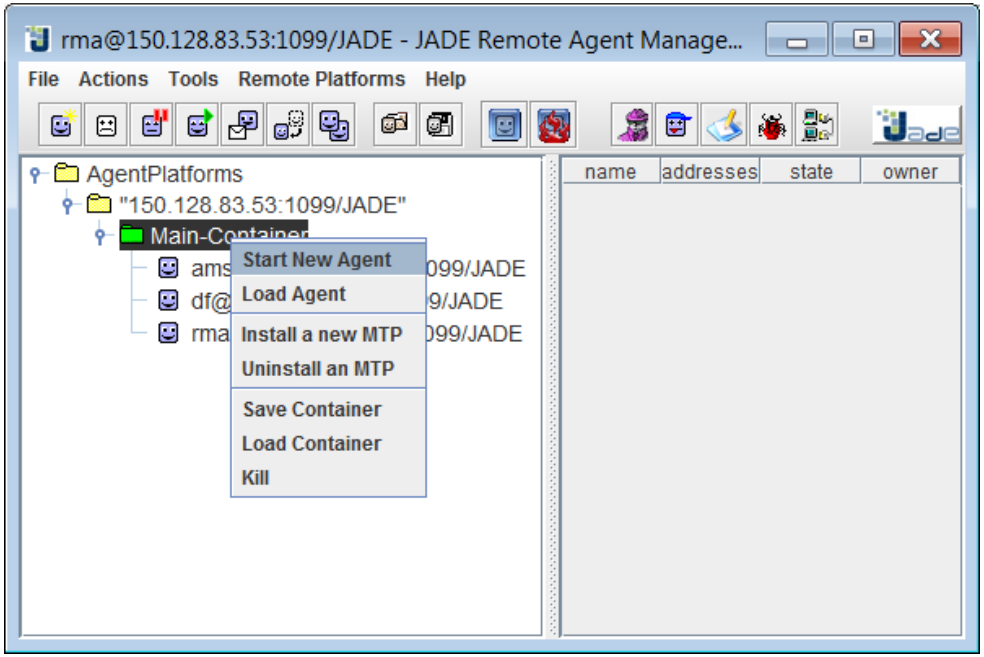
Ejecutando agentes

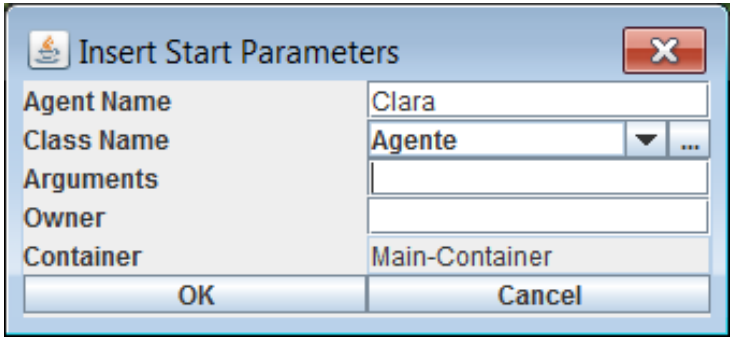
- La plataforma Jade permite ejecutar opcionalmente una interfaz gráfica que proporciona herramientas para depurar nuestros agentes.



Ejecutando agentes

- También se pueden lanzar desde el interfaz:





Ejecutando agentes

- Finalización de agentes
  - Un agente cuando finaliza su ejecución ejecuta su método *doDelete()*
  - *doDelete()* llama al método *takeDown()* que se puede redefinir para incluir las operaciones relativas a la limpieza de elementos del agente (por ejemplo, cerrar una base de datos, borrarse del servicio de páginas amarillas, etc.)
- Paso de argumentos a un agente
  - El método *getArguments()* devuelve un objeto de la clase *Object[]*
  - **EJERCICIO**: Modifica Agente para que reciba como argumento el nombre de un fichero e imprima su contenido por pantalla en el método *setup()*

    http://chuwiki.chuidiang.org/index.php?title=Lectura_y_Escritura_de_Ficheros_en_Java

Programando agentes

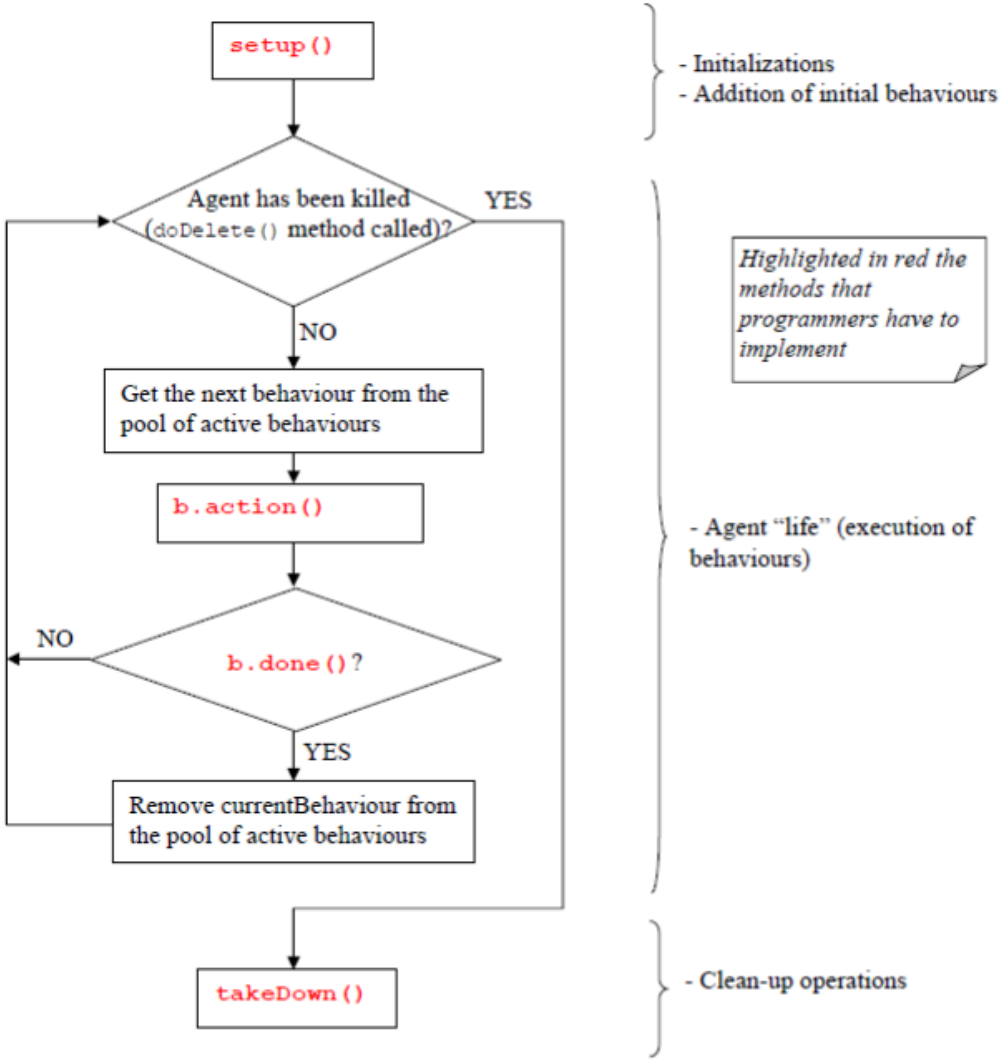
```
import jade.core.Agent;

public class Agente extends Agent {
   public void setup() {
      Object[] args = getArguments();
      if (args == null) {
         System.out.println("Falta el nombre del fichero");
      } else {
         File f = new File((String) args[0]);
         if (!f.exists()) {  System.out.println("El fichero no existe.");
         } else {
                try {
                   FileReader stream = new FileReader(f);
                   BufferedReader buffer = new BufferedReader(stream);
                   String linea;
                   while ((linea = buffer.readLine()) != null) {
                         System.out.println(linea);
                    }
                   entrada.close();
                } catch (Exception e) { e.printStackTrace();  }
         }
      }
   }
}
```

Especificando tareas: Comportamientos

- La clase jade.core.behaviours.Behaviour proporciona los comportamientos (funciones) que el agente ejecutará.
- Para que un agente ejecute una tarea hay que crear una instancia del comportamiento adecuado y llamarla con el método: *addBehaviour(comportamiento)*
- Los comportamientos han de implementar dos métodos:
  - *action()* que contiene el código con las tareas a realizar en ese comportamiento.
  - *done()* que es la condición que indica si el comportamiento ha terminado o no de ejecutarse.

Ciclo de vida de un agente



Ciclo de vida de un agente

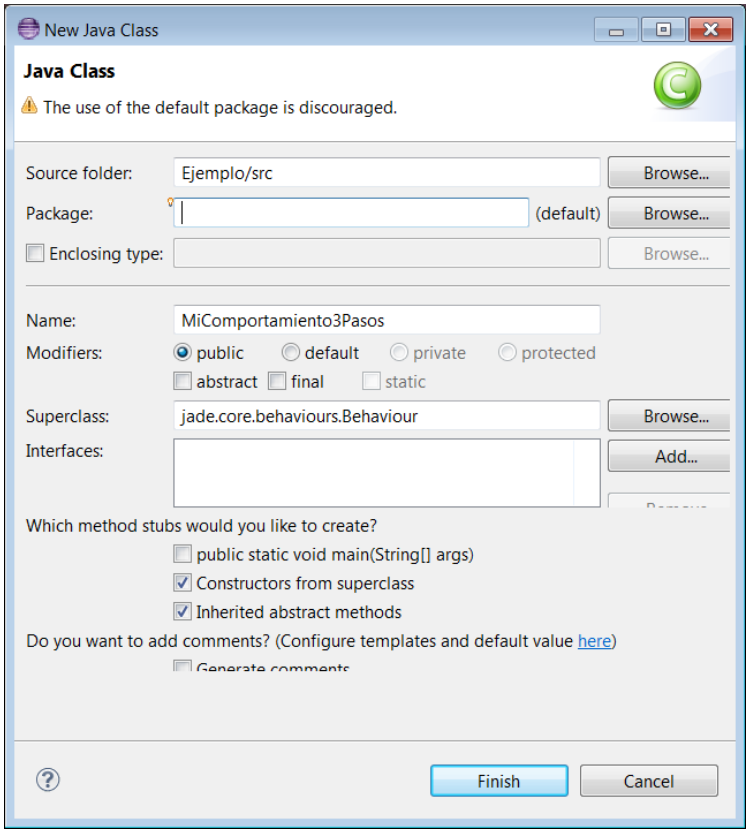
- Tipos de comportamientos: Se pueden agrupar los comportamientos de Jade en tres grandes grupos:
  - **One-shot**: Se ejecutan una sola vez.
  - **Cíclicos**: Nunca son eliminados del conjunto (pool) de comportamientos del agente y su método *action()* siempre utiliza el mismo código. Nunca finalizan.
  - **Genéricos**: Más genéricos, el código que se ejecuta depende del estado del agente y, eventualmente, finalizan su ejecución.

Ejemplo de comportamiento One-shot

```
public void action() {
    System.out.println("Hola, soy el agente " + getAID().getLocalName());
    System.out.println("Mi identificador completo es " + getAID().getName());
    System.out.println("Mis direcciones son:");
    Iterator it = getAID().getAllAddresses();
    while (it.hasNext()) {
        System.out.println(" ---> " + it.next());
     }
}
```

- **EJERCICIO**: incorpora un comportamiento one-shot para que la lectura del fichero se haga dentro del comportamiento.

Ejemplo de comportamiento genérico




```
public boolean done() {
    return step == 3;
}
```

```
public class MiComportamiento3Pasos
        extends Behaviour {
  private int step = 0;
  public void action() {
    switch (step) {
      case 0:
        // Realiza la operación X
        step++;
        break;
      case 1:
        // Realiza la operación Y
        step++;
        break;
      case 2:
        // Realiza la operación Z
        step++;
    }
  }
}
```

Otros comportamientos

- WakerBehaviour y TicketBehaviour
  - Su ejecución es diferida.
  - Se ejecutan cuando se ha cumplido un determinado tiempo.
  - WakerBehaviour es un comportamiento one-shot.
  - TicketBehaviour es un comportamiento cíclico.

```
public class MiDespertador extends WakerBehaviour {
    public MiDespertador( Agent a, long t) {
        super(a, t);
    }
    public void onWake() {
        System.out.println("Hola, al fin me desperté!!!");
    }
}
```

```
public class Cansino extends TickerBehaviour {
    public Cansino( Agent a, long t) {
        super(a, t);
    }

    public void onTick() {
        System.out.println("Hola, insisto: Buenos días");
    }
}
```

Ejercicio 1

- El análisis de un gráfico debe pasar por 4 fases secuenciales:
  - Pre procesado de la información.
  - Análisis del gráfico.
  - Validación de resultados.
  - Almacenamiento de resultados.
- Crea un agente con un comportamiento compuesto que pueda implementar este análisis.
- El agente recibe como parámetro el nombre del fichero que contiene el gráfico.
- En cada fase el agente debe:
  - Pre procesado: Leer el fichero e imprimirlo por pantalla.
  - Análisis: sólo hay que imprimir el mensaje "Analizando"
  - Validación: Solicitar al usuario por teclado si valida o no el análisis. Si no lo valida por 3 veces debe almacenar un resultado negativo.
  - Almacenamiento: Imprimir un mensaje diciendo si ha finalizado con éxito o no.

Ejercicio 2

- Escribir un agente JADE que tenga tres comportamientos. En el setup() el agente ha de mostrar en consola el tiempo inicial mediante el uso de la clase java.util Date y cargar los tres comportamientos que se describen a continuación:
  - El primer comportamiento es un comportamiento genérico que realiza 4 pasos y en cada paso ha de calcular si los 4 números pasados como parámetros al agente son primos o no.
  - El segundo comportamiento es un comportamiento waker que al pasar 10 segundos imprimirá el mensaje "Me acabo de despertar y son las …" donde … será la fecha actual.
  - El tercer comportamiento es un comportamiento cíclico que cada 3 segundos imprimirá un mensaje indicando cuantas veces se ha ejecutado hasta cada momento junto con la fecha actual.

Ejercicio 3

- Escribir un agente JADE que lance otros cuatro agentes y cada uno ellos tenga un comportamiento que muestre por la consola su nombre y la fecha en que se ha ejecutado (consultar la API de Jade y búsquedas de Google para aprender a realizar este ejercicio).