Introducción

Crear el primer Agente

El objetivo de esta primera práctica es una primera toma de contacto con el sistema multi-agente que se utilizará a lo largo de la asignatura.

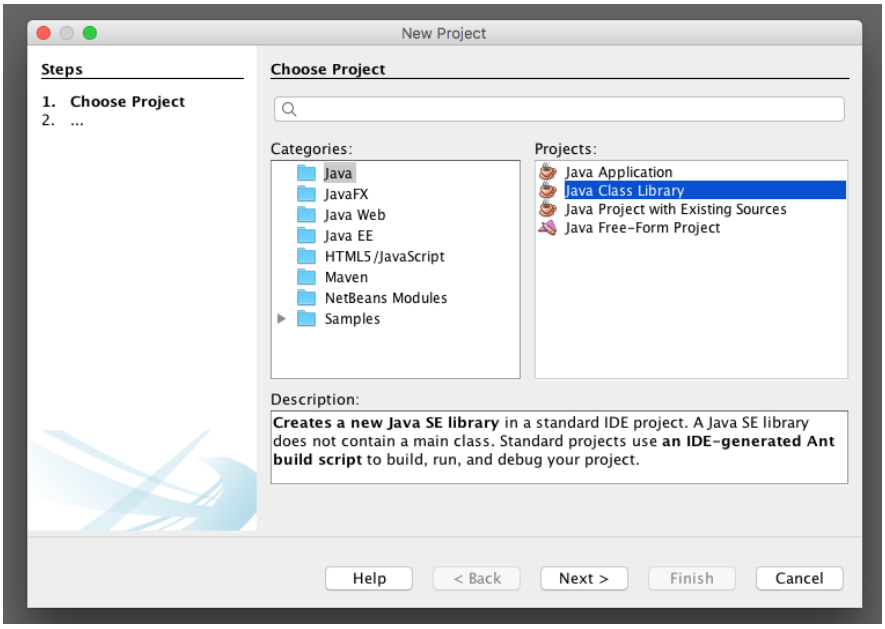
Los **Objetivos** de esta práctica son los siguientes:

- Presentar la estructura que debe tener un proyecto **NetBeans** que utilizaremos a lo largo del desarrollo de las prácticas.
- Presentar la estructura general que deberá tener una clase que represente un agente en el desarrollo de las prácticas.
- Por último, la creación de un primer agente para tomar contacto con la biblioteca multi-agente que se utilizará.

1. Creación de un proyecto NetBeans

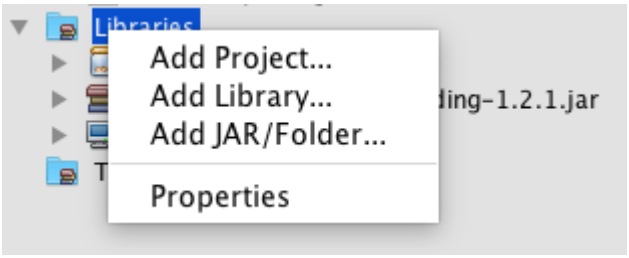
Vamos a presentar la estructura general que deberán tener todos los proyectos que se crearán para las prácticas. El entorno de desarrollo elegido es NetBeans 8.2, aunque será válido siempre que sea NetBeans 8.x. No estará permitido otro entorno de desarrollo en las prácticas. Además debemos asegurarnos que la versión de Java sea la 1.8.

Mediante el asistente seleccionaremos la opción de creación de un proyecto para una biblioteca Java:



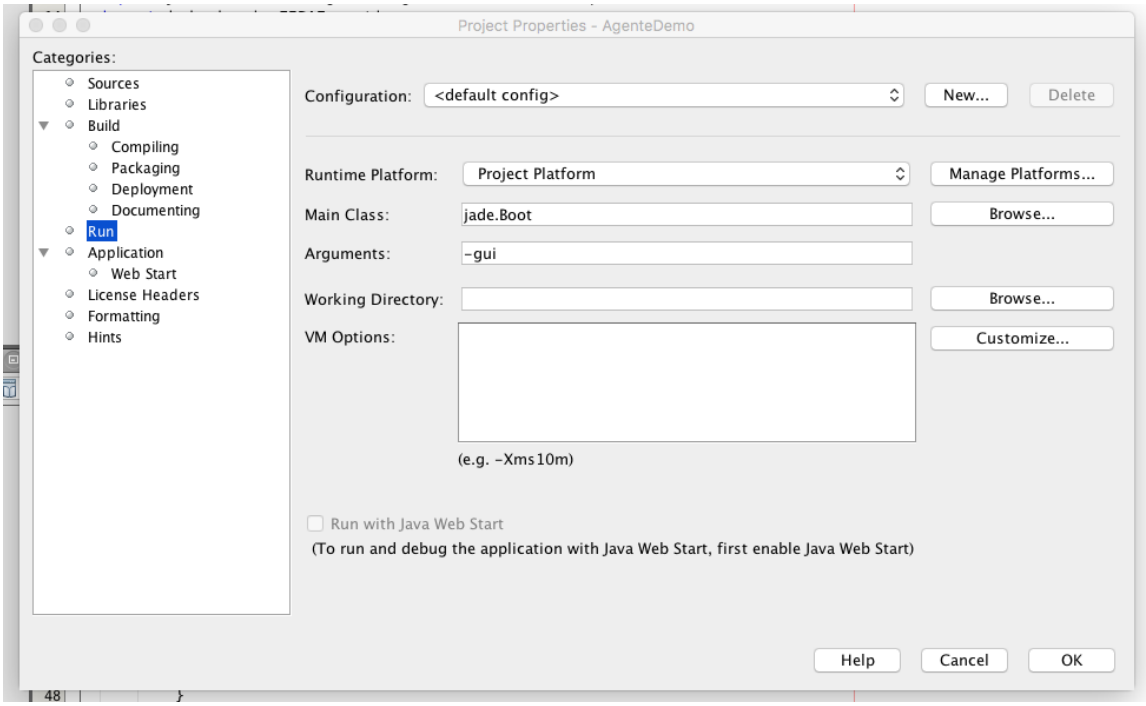
Seleccionaremos un nombre representativo para nuestro proyecto Prueba y completamos la configuración del proyecto asegurándonos que se crea la carpeta "./lib" donde se incluirán las bibliotecas necesarias para el desarrollo del proyecto que no sean las propias de Java.

Una vez finalizada la configuración del proyecto debemos asegurar la inclusión de la biblioteca de la plataforma multi-agente ("jade.jar") en el proyecto. Para ello deberemos ir a la carpeta de bibliotecas del proyecto, y con el botón derecho del ratón, seleccionar la inclusión del fichero ".jar". Localizamos el archivo "jade.jar" y seleccionamos la opción de copiarlo al directorio "./lib" de nuestro proyecto.



De esta forma ya tendremos configuradas las bibliotecas necesarias para el desarrollo de nuestro proyecto.

El siguiente paso es configurar la ejecución de nuestro proyecto para poder realizar las pruebas necesarias del mismo. Para ello seleccionamos las propiedades del proyecto y localizamos la de ejecución. Configuramos esta propiedad con las opciones que se muestran en la imagen.



Para finalizar la configuración de nuestro proyecto debemos crear los paquetes necesarios donde se incluirán las clases que permitan resolver el problema que tengamos planteado. Como norma general deberemos necesitar, para esta primera práctica, los siguientes paquetes:

- Paquete `agentes`: en este paquete incluimos todas las clases que van a representar los agentes presentes en la solución de nuestro problema.
- Paquete `GUI`: en este paquete incluimos todas las clases que representarán la interfaz gráfica que tenga nuestro proyecto. Estas interfaces serán la visualización que tengamos de nuestros agentes dentro del sistema.

De esta forma ya tenemos el proyecto NetBeans finalizado y podremos empezar a incluir las clases necesarias que nos permitan resolver nuestro problema. Una consideración final, es muy recomendable que se utilice el sistema **Git**, con el que ya habéis trabajado en asignaturas anteriores, como sistema de control de versiones en vuestros proyectos. De esta forma se tiene un desarrollo más controlado de vuestros proyectos y también os ayudará en el desarrollo de las prácticas en equipo.

2. Estructura para una Clase de Agente

Ahora se presenta la estructura general que deberá tener una clase que representará nuestro agente en el proyecto. Esta estructura se utilizará como punto de partida y posteriormente deberéis personalizarla para adaptarla al propósito que deba tener nuestro agente en el sistema.

Como norma de estilo, todas las clases que representen a un agente deberán estar incluidas en el paquete `agentes` de vuestro proyecto y siempre empezarán por la palabra `Agente`. En el ejemplo que nos ocupa será `AgenteEsqueleto`, que representa la estructura que deberán tener cada uno de los agentes que creemos en nuestros proyectos.

El archivo de esta clase os lo podéis descargar desde ILIAS y así no será necesario el asistente para la creación de una clase Java. Ahora pasaré a explicar las diferentes partes que lo componen:

Declaración del Agente

```java
/**
 *
 * @author pedroj
 * Esqueleto de agente para la estructura general que deben tener todos los
 * agentes
 */
public class AgenteEsqueleto extends Agent {
    //Variables del agente
```

Una vez que hemos definido nuestro agente incluimos las variables que sean necesarias, recordar que las variables deben ser privadas. Otra particularidad que debemos recordar es que no será necesario declarar ningún constructor. Solo debemos sobreescribir los métodos `setup()` y `takeDown()`.

Método `setup()`

Este el el método donde estará la inicialización del agente y la inclusión de las tareas principales del mismo.

```
@Override
protected void setup() {
   //Inicialización de las variables del agente

   //Configuración del GUI

   //Registro del agente en las Páginas Amarillas

   //Registro de la Ontología

   System.out.println("Se inicia la ejecución del agente: " + this.getName());
   //Añadir las tareas principales
}
```

En la plantilla se describen el orden que deberá seguir los distintos elementos presentes en el método. No todos los elementos estarán presentes en todos los agentes que creemos al principio, pero sí estarán en los agentes que forman parte de las prácticas obligatorias que se deberán entregar.

El registro en las páginas amarillas será necesario para aquellos agentes que proveen de servicios a otros agentes. Los agentes que no estén en ese grupo no es obligatorio su registro.

Para añadir una tarea que el agente deba completar se invocará el método `addBehaviour( Behaviour )`. De esta forma la tarea se añadirá a las que deba hacer el agente y su ejecución se realiza en una arquitectura **single thread.** No se tendrá que tener cuidado por las secciones críticas pero sí para la sincronización.

Método `takeDown()`

El método será invocado cuando el agente finalice su ejecución o el programado invoque el método `doDelete()` de forma explícita.

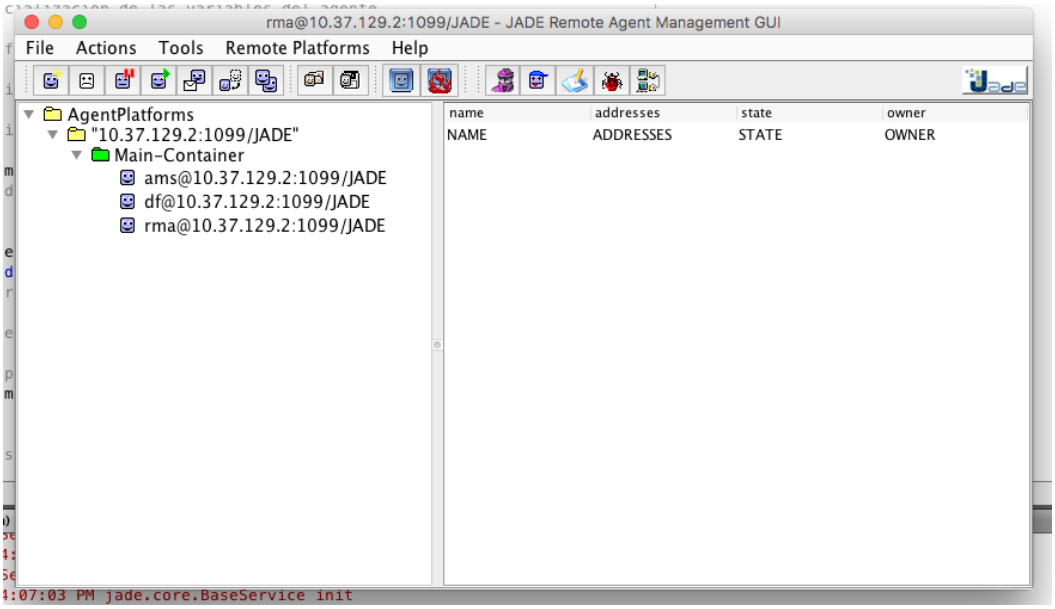
```
@Override
protected void takeDown() {
   //Desregristo del agente de las Páginas Amarillas

   //Liberación de recursos, incluido el GUI

   //Despedida
   System.out.println("Finaliza la ejecución del agente: " + this.getName());
}
```

En la imagen se muestra las acciones que deberán hacerse en este método, son todas aquellas necesarias para que la finalización del agente se realice de forma ordenada y sin dejar recursos pendientes en el sistema.

Para completar la clase del agente se incluirán los métodos propios para su normal funcionamiento, incluyendo los métodos de acceso para los atributos definidos previamente. Por último, se incluirán las clases internas que representan las tareas propias que el agente deberá llevar a cabo. Más adelante se mostrará un ejemplo relativo y en posteriores guiones veremos más claramente lo que quiero decir.

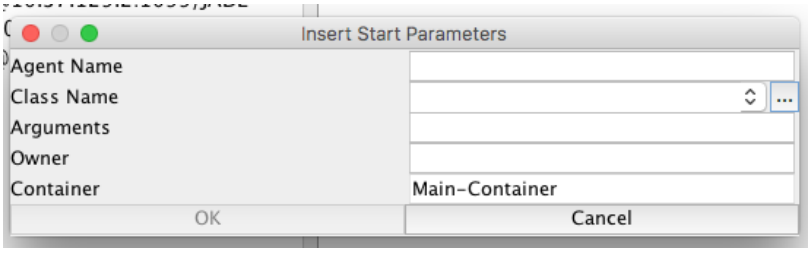
Ejecución de un agente

Si ejecutamos el proyecto deberá aparecer en pantalla la siguiente imágen:

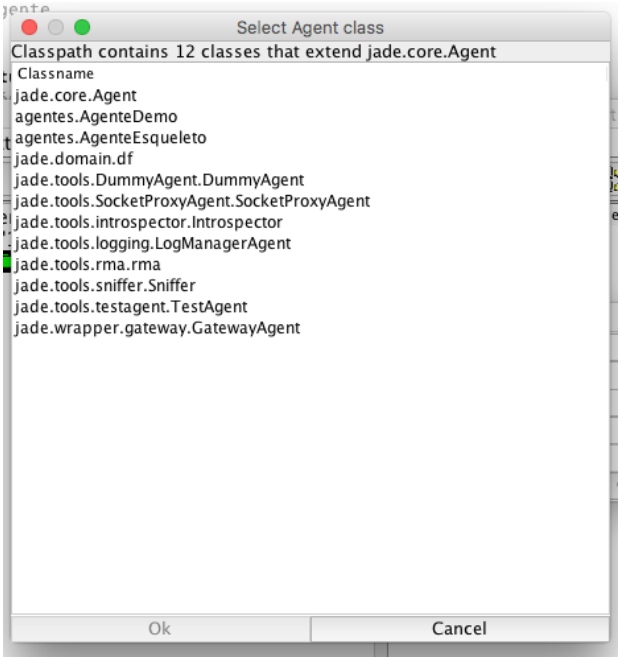


Lo que se muestra en la imágen es el agente **rma** que presenta una interfaz gráfica para la plataforma de agentes. Este agente está presente únicamente en el contenedor principal de la plataforma y de esta forma podremos trabajar con ella de una manera más cómoda.

Si queremos ejecutar un agente, debemos seleccionar el contenedor donde lanzarlo y pulsar en el icono para crear un agente:



En la figura anterior vemos la ventana que se crea para la creación del agente. Debemos elegir un nombre y la clase a la que pertenece al agente. Para ello podemos pulsar en el botón desplegable que nos mostrará todas las clases de agente que conoce nuestro proyecto.



Si completamos todos los pasos de forma apropiada se mostrará en el contenedor nuestro nuevo agente. Dependiendo de los elementos presentes en el agente se podrá tener una visión gráfica o no del mismo. En la última parte del guión se mostrará un agente que tiene una interfaz gráfica y además realizará tareas que se mostrarán en la misma.

Prueba práctica

Probar si la ejecución del `AgenteEsqueleto` produce algún resultado. Modificar el resultado para que solo muestre su nombre y no lo que aparece originalmente.

3. Creación del primer agente y tarea

Se crea un agente que tiene el siguiente comportamiento:

1. Muestra una ventana que servirá para mostrar los mensajes del agente. Además tendrá un botón que finalizará el agente mediante un cuadro de diálogo para confirmarlo.
2. Registrar el agente en servicio de páginas amarillas.
3. Crear una tarea para el agente que sea periódica y que se repita cada 10 segundos. La acción de la tarea será incrementar un contador del agente que mostrará el número de veces que se ha ejecutado la tarea.

Creamos el proyecto NetBeans cuyo nombre será AgenteDemo, siguiendo las indicaciones que se han presentado previamente en este guión. Una vez que se ha finalizado con la configuración del proyecto se crean dos paquetes:

- Un paquete `GUI`: Donde se incluirán dos clases: una para presentar la interfaz del agente y otra con el cuadro de diálogo para confirmar la finalización del mismo.
- Un paquete `agentes`: En este paquete se creará la clase AgenteDemo que contiene la implementación de nuestro agente.

Antes de pasar con la presentación del agente se mostrarán las dos clases del paquete `GUI` con aquellos elementos que son los más interesantes.

Clase AgenteDemoJFrame

Mediante el asistente creamos una clase **JFrame** con el nombre de esta sección. Realizamos el diseño gráfico mediante el botón de diseño. La ventana contendrá los siguientes elementos:

- **JTextArea**: Para presentar los mensajes del agente asociado.
- **JButton**: Para lanzar el cuadro de diálogo que nos permitirá finalizar con la ejecución del agente.

El diseño concreto dependerá de las necesidades propias y queda libre, en el ejemplo se presenta uno posible.

Los elementos de personalización son los siguientes atributos privados:

- Un atributo que nos permitirá enlazar el **JFrame** con su agente.
- Un atributo para crear el cuadro de diálogo.

El constructor deberá dar valor al atributo que nos permitirá enlazar con el agente y también cambiaremos el título con el nombre del agente para que se tenga una visualización clara que esa interfaz corresponde a ese agente.

Para poder incluir un mensaje en la interfaz se debe crear un método público para que lo pueda invocar el agente y así presentar los mensajes por el **JFrame.** Para finalizar debemos crear el método de acción del botón y capturar la finalización del **JFrame** para crear el cuadro de diálogo que pondrá fin a la ejecución del agente.

Clase FinalizacionDialog

Esta clase la crearemos mediante el asistente de **JDialog** y realizaremos un diseño donde presentaremos dos **JButton**, uno para confirmar la finalización de la ejecución del agente y otro para cancelar el cuadro de diálogo.

Esta clase deberá disponer de un atributo privado para saber el agente que tiene asociado y así poder finalizar con su ejecución.

El constructor deberá inicializar este atributo y asegurarnos que se crea de forma modal, así el usuario deberá centrarse en este cuadro de diálogo y no interactuar con la interfaz del agente de ninguna otra forma.

El botón de cancelación o la finalización del cuadro de diálogo no deben finalizar la ejecución del agente. Solo en el botón de finalización invocamos el método `doDelete()` del agente, que forzará la finalización del mismo.

Clase AgenteDemo

Esta clase la creamos en el paquete `agentes` y tendrá la implementación de las acciones que se han descrito anteriormente.

```
/**
 *
 * @author pedroj
 */
public class AgenteDemo extends Agent {
    private AgenteDemoJFrame myGui;
    private int ejecuciones;
```

El agente necesita de dos atributos, uno para llevar el número de ejecuciones y otro para tener acceso a su interfaz gráfica. Ambos atributos serán privados para que solo el agente tenga acceso a ellos.

Método `setup()`

La inicialización del agente implica la creación de la interfaz y la presentación en ella del mensaje de inicio. Además se inicializa el atributo de ejecuciones.

```
//Configuración del GUI y presentación
myGui = new AgenteDemoJFrame(this);
myGui.setVisible(true);
myGui.presentarSalida("Se inicia la ejecución de " + this.getName() + "\n");

//Incialización de variables
ejecuciones = 0;
```

El siguiente paso es el registro en el servicio de páginas amarillas, en case se realizará una explicación más detallada de qué elementos son importantes en este paso.

La finalización del método es con la creación de la tarea cíclica que debe realizar el agente.

Método takeDown()

Al igual que el proceso de registro hay que llevar el proceso de baja en las páginas amarillas. Se debe realizar en este método y en clase se dará una explicación más detallada del mismo.

El único recurso que tiene nuestro agente es su interfaz que deberemos liberar. Para finalizar presentaremos el mensaje de despedida en la consola. Como ya no disponemos de interfaz hay que realizarlo allí. Además, es mejor hacerlo así para que tengamos constancia que nuestro agente ha finalizado, de otro modo el mensaje se perdería.

Al tratarse de un agente bastante simple no tiene necesidad de métodos auxiliares para su funcionamiento.

Como la única tarea que realiza nuestro agente es propia la implementaremos como una clase interna. La ventaja es que tendremos acceso a los atributos de nuestro agente. Si fuera una tarea compartida entre más de un agente, lo propio sería crear un paquete de tareas donde se incluirían las clases que representan estas tareas.

Clase TareaEjemplo

Como la tarea será cíclica pero se ejecutará con una periodicidad deberemos heredar de la clase **TickerBehaviour**.

```
//Clases que representan las tareas del agente
public class TareaEjemplo extends TickerBehaviour {
    //Tarea de ejemplo que se repite cada 10 segundos
    public TareaEjemplo(Agent a, long period) {
        super(a, period);
    }
```

De esta clase solo debemos sobreescribir el método `action()` donde se indicará la acción a realizar cuando se haya cumplido el periodo. En nuestro caso incrementar el atributo que lleva la cuenta de las ejecuciones y presentar un mensaje en la interfaz que dejará constancia de la ejecución.

```
@Override
protected void onTick() {
    ejecuciones++;
    myGui.presentarSalida("\nEjecución número: " + ejecuciones);
}
```

De esta forma queda completo el proyecto con las especificaciones que se habían pedido del mismo.

Ejercicio práctico

Realizar una modificación en la interfaz para que no se pueda finalizar la ejecución del agente hasta que no haya completado, al menos, 4 ejecuciones de la tarea **TareaEjemplo.**

Crear una nueva tarea al agente que realice lo siguiente:

- Cada vez que se complete una tarea cuyo número de ejecución sea par: muestre una secuencia de números, tantos como ejecuciones se hayan alcanzado.
- Cada vez que se complete una tarea cuyo número de ejecución se impar: muestre la secuencia de fibonacci, tantos como ejecuciones se hayan alcanzado.
- Cuando se alcancen las 10 ejecuciones finalizará esta tarea.