2014

الوجيز في البرمجة متعددة العملاء على المنصة jade

زكرياء اللبي

يعد هذا الكتيّب مقدمة لمن يريد احتراف البرمجة متعددة العملاء على المنصة JADE

بسم الله والحمد لله والصلاة والسلام على رسول الله

أما بعد...

عزيزي القارئ والمتعلم، أقدم لك هذا الكتيّب والعمل المتواضع ليكون لك كنقطة على السطر تكون منطلقا لك للغوص في أحد مجالات المعلوماتية الواسعة وهو مجال الأنظمة المتعددة العملاء.

أقدم هذا العمل المتواضع إسهاما في إثراء المكتبة الرقمية العربية، اذ يظهر جليا افتقارها الى الكتب المنهجية المتخصصة في مجال المعلوماتية.

عزيزي القارئ، إن هذا الكتيّب يقدم لك فكرة نظرية مبسطة عن هذا المجال، ثمّ فكرة عملية تطبيقية لكيفية انجاز بعض المشاريع على احدى المنصات المختصة ببرمجة وتسيير الأنظمة المتعددة العملاء.

عزيزي القارئ، إن هذا الكتيّب يقدم لك رأس الخيط فقط، فلا تتوقف عند ما جاء به وإنما اجعله كعتبة دخولك الى مجال واسع شاسع لتنطلق بحثا عن المزيد حتى الاحتراف.

أخيرا، أرجوا من المولى سبحانه وتعالى أن يجعل هذا العمل خالصا لوجهه الكريم ثم أطلب منك دعاءً صالحا بظهر الغيب.

زكرياء اللبي

لإبداء آرائكم وملاحظاتكم

zakaria.lab@gmail.com

الفهرس:

العميل والانظمة متعددة العملاء

L'agent et les systemes multi-agents (SMA)

البرمجة على منصة jade

La programmation sur la plateforme JADE

العميل والانظمة متعددة العملاء

L'agent et les systemes multi-agents (SMA)

عزيزي القارئ، إن هذا الفصل يقدم لك فكرة عامة عن العملاء والأنظمة متعددة العملاء وخصائصيهما.

إن هذه المقدمة النظرية المبسطة لا تكفيك للإحاطة التامة بجوانب هذا المجال، ولكن تُسطّر لك الخطوط العريضة التي يقوم عليها بصفة مبسطة وميسرة. عليك بالمزيد من البحث لمزيد من الإطلاع.

1. العميل:

لا يوجد تعريف وحيد متوافق عليه للعميل وانما كلٌّ يعرفه على حسب وجهة نظره ومفهومه، فيما يلي بعض التعريفات لماهية العميل:

Shoham : Un agent est une entité qui fonctionne continuellement et de manière **autonome** dans un environnement où **d'autres processus** se déroulent et **d'autres agents** existent."

العميل هو كيان يعمل بشكل مستمر ومستقل في بيئة معينة في وجود عمليات الأخرى وعملاء اخرين.

Russell: Un agent est une entité qui **perçoit son environnement** et **agisse** sur celui-ci

العميل هو كيان يستوعب بيئته ويعمل فيها

Wooldrige et Jennings: Un agent est un système informatique, **situé** dans un environnement, et qui agit d'une façon **autonome** pour atteindre les **objectifs** (buts) pour lesquels il a été conçu

العميل هو نظام الكمبيوتر متواجد في بيئة ما، ويعمل بشكل مستقل لتحقيق الأهداف التي صمم من أجلها

Ferber : Un agent est **une entité autonome**, réelle ou abstraite, qui est capable **d'agir** sur elle-même et sur son environnement, qui, dans un **univers multi-agents**, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents

العميل هو كيان مستقل حقيقي أو مجرد وهو قادر على التصرف في ذاته وبيئته، وفي حالة بيئة متعددة العملاء يمكنه التواصل مع غيره من العملاء، ويكون سلوكه بناء على ملاحظاته والمعرفة والتفاعل مع العوامل الأخرى

ومن خلال منطلق فكرة العميل والانظمة متعدد العملاء والتي بنيت على اساس محاكات عمل وتصرفات البشر في انجاز مهامهم، وفي كنف التعريفين السابقين فإننا نستطيع استخلاص التعريف التالي :

العميل هو عبارة كيان برمجي صمم ليتواجد في ببيئة معينة ولتحقيق اهداف محدد، والعميل له القدرة على العمل باستقلالية وبناءً على معارف يمتلكها إما بشكل مسبق او يتعلمها من خلال التعامل والتفاعل والتواصل مع بيئته او عملاء اخرين، كأن يغير في بيئته أو يتعاون مع عملاء اخرين لإنجاز مهام مشتركة.

2. خصائص العميل:

تتلخص الخصائص العامة للعميل في ما يمكنه من العمل المستقل، حيث يمكننا ان نستخلص هذه الخصائص من طريقة وتركيبة عمل الانسان في ما يخص التعامل مع المشاكل والاحداث التي تحدث في الوسط او البيئة المتواجد فيها وكذا طريقة اتخاذه لقراراته.

وتتركز اهم الخصائص في النقاط التالية:

- **التموضع(situé)**: كل عميل يتواجدا في اساسا في بيئة معينة، وله القدرة على التعامل والتفاعل مع بيئته عن طريق مدخلات ومعطيات يتحصل عنها انطلاقا من بيئته نفسها.
- **الاستقلالية(autonome)**: للعميل استقلالية تامة في ما يخص ذاته واتخاذ قراراته وكيفية تعامله مع الاحداث والعملاء الاخرين المتواجدين معه في نفس البيئة، فهو يتحكم في تصرفاته بدون تدخل طرف اخر، وانما يتخذ قراراته بناءً على حالته الآنية وباستقلالية تامة.
- **الاستباقية (proactif)**: للعميل القدرة على أخذ زمام المبادرة في الوقت اللازم والمناسب، بناءً على ما يخدم تحقيقه لأهدافه، أي ان له سلوك انتهازي ان صحّ التعبير.

❖ **الاستجابة المباشرة(répondre à temps)**: يجب ان يكون العميل قادرا على الاستجابة الآنية للتطورات والاحداث التي تقع في محيطه، وهذا يتطلب منه ان يبدي الاستجابة والتفاعل في الوقت والمدة المطلوبة (Temps réel).

❖ **اجتماعي(social)**: للعميل خصائص اجتماعية تتمثل اجمالا في التواصل والتعامل مع العملاء الاخرين الذين يشاركونه بيئته أو الانسان وايضا الهجرة من بيئة الى اخرى في بعض الحالات.

3. أقسام العملاء:

هناك عدة طرق لتصنيف العملاء، ولكن عادة ما نحكم على العميل انطلاقا من طريقة اتخاذه لقراراته و تفاعله وتأثيره في بيئته والاحداث الجارية فيها، حيث من جهة يتحسس واقعه عن طريق اللاقطات(Capteurs) –برمجية او فيزيائية– ومن جهة اخرى يؤثر فيه عن طريق مسنداته(Effecteurs) –برمجية او فيزيائية– والمقصود بالمسندات هي الآلية او الاداة التي من خلالها يحدث التغيير في بيئته كذراع الآلي مثلا.

تلخيصا للفقرة السابقة فإن تصنيف العميل يعتمد على طريقة ربط معارفه مه حركاته وقراراته.

وقد قسّم الكاتبان Russel et Norvig الى اربع اقسام، سوف نذكر هاته الاقسام مجمعة في مجموعتين اساسيتين بناءً على إمكانية التفكير لدى العميل كالتالي:

أ– العميل التفاعلي(Agents réactifs):

وعميل لا يفكر ولا يخطط وانما يستجيب للتغيرات الحاصلة في بيئته، حيث يستخلص المعلومات عن طريق اللاقطات ويتفاعل معها على اساس قواعد معرفة مسبقا مطبقا لها عن طريق المسندات بناءً على قواعد مسبقة مخزنة لديه، وتضم هذه المجموعة قسمان:

❖**عميل ذو رد فعل بسيط(à réflexes simples)**: هذا النوع يتفاعل فقط مع الاحداث الآنية بناءً على قواعد معرفة مسبقا بنيتها بهذا الشكل:

إذا تحقق (الشرط) إذن (الحدث)

Si (condition) alors (action)

اي انه يطبق الحدث الموافق للقاعدة تحققَ شرطها في الوقت الآني، ومما يميز هذا النوع من العملاء هي سرعته الكبيرة في التفاعل مع الاحداث، وكمثال على ذلك نظام الحماية في البارجة الحربية الذي وبمجرد دخول طائرة معادية الى مدى معين في اتجاهها يطلق وراءها صاروخ(اذا دخلت طائرة غريبة إذن ارسل وراءها صاروخ).

❖**عميل يتتبع مسار العالم(conservant une trace du monde)**: في المثال السابق كان تصرف العميل مقتصرا فقط على الحالة الآنية التي كان فيها، فرد بإطلاق النار على التهديد عند وجوده، لكن لو اعتبرنا وجود حاجز تعذر بوجوده رصد التهديد فان هذا لا يعني مطلقا عدم وجود التهديد، فان اختفت الطائرة بفعل الحاجز بعد رصد العميل لها لا يعني انتهاء التهديد، لذا وجب على العميل ان يأخذ يعين الاعتبار التضاريس، بمعنى اخر يجب على العميل ان يكون ملمّا ببيئته، فالمقصود بالعالم هنا هو البيئة.

ان هذا النوع من العملاء يقوم دائما بتحديث معلوماته عن بيئته على اساس نوعين ومن المعلومات، الاولى بناءً على كيفية تطور العالم من حوله، مثلا اذا كانت سرعة الطائرة 30 كلم/سا وعلى بعد 50كلم فانها ستكون بعد 5د على بعد 30كلم عن البارجة، وثانيا بناءً على تغييره هو في العالم، مثلا اذا قامت البارجة بالاستدراة فانه يعلم ان كل شيء سيتغير بناءً على ذلك.

ب- العميل المعرفي(Agents cognitifs):

كلمة "معرفي" هي ترجمة حرفية حيث لم اجد ترجمة مناسبة حقيقةً، والمعنى انه يعتمد على معارفه ان صح التعبير، وهذا النوع من العملاء يتميز بانه يقوم بعدة عمليات قبل اتخاذ قراراته على اساس اهدافه.

❖**عميل ذو اهداف(ayant des buts)**: في كثير من الاحيان يتطلب اتخاذ القرار اكثر من مجرد الاحاطة بالبيئة، فمثلا لتحديد الاتجاه الذي ينبغي ان تسير فيه البارجة لا يكفي مجرد معرفة محيطها، لأنه ببساطة يتطلب منها الالمام بالمحطة المنشودة.

ان هذا العميل يجمع بين الوصف الجيد لمحيطه ولهدفه لاتخاذ قراراته، حيث انه يحاول ان يكون نظرة استباقية للمستقبل، فيطرح على نفسه الاسئلة التالية: ماذا سيحدث اذا فعلت هذا الامر او هذا الامر؟ هل سترضيني هذه النتيجة؟.

من جهة اخرى فان العمليات التي ينفذها هذا النوع من العملاء قبل اتخاذه لقراراته تجعله ابطء من القسم الاول من العملاء.

❖عميل يستخدم دالة المنفعة(utilisant une fonction d'utilité): في عديد الاحيان لا يكفينا الالمام بالواقع والاهداف لاتخاذ الاقرارات الانسب، فمثلا عندما تتعدد المسارات المؤدية لنفس الهدف فأيها نختار؟

ان هذا النوع من العملاء يعتمد على دوال تسمى: "Les fonctions d'utilité" اي دوال المنفعة، وهي دوال تحدد اي القرارات افضل، مثلا اذا كان هناك اكثر من طريق للوصول البيت فهذه الدالة تمكننا من تحديد الاقصر مثلا او الاقل خطورة او الاقل تكاليف وهكذا.

4. الانظمة المتعددة العملاء(Systèmes multiagents):

لقد تكلمنا في كل ما سبق عن العميل بشكله المنفرد، ولكن لا يتواجد العميل منفردا في الحقيقة، بل يكون محاطا بعملاء اخرين مكونين بذلك نظام يدى بمتعدد العملاء "Systèmes multiagents" ويرمز له اختصارا بـ SMA، ومن اهم خصائصه:

- كل عميل يمتلك معلومات غير كاملة، اي ان نظرته محدودة.
- لا يوجد تحكم كامل في النظام.
- البيانات غير مركزية.
- عدم وجود تزامن في تطبيق الاوامر.

5. خصائص الانظمة متعددة العملاء:

أ- التفاعل(Interactions):

مستوى تفاعلي عالي في ما يخص التنسيق والتواصل والتنظيم والحل المشترك للمشاكل.

ب- التنسيق(Coordination):

هناك ثلاث اقسام للعملاء من ناحية التنسيق بينها، القسم الاول العملاء المتعاونون(agents coopératifs)، وكما يظهر من الاسم فان عملاء هذا القسم يتعاونون مع العملاء الاخرين لأداء مهامهم، اما الثاني فهو العملاء المنفردين(agents individualistes)، ويضم العملاء الذين يعتمدون على انفسهم فقط للأداء مهامهم، القسم الثالث هم العملاء المتنافسون(agents compétition)، ويضم الصفة الباقية وهو التنافس بين العملاء.

للإشارة فإن التنسيق يتم عن طريق اما التفاوض(négociation) او التحالف(coalitions).

ت- التواصل(Communication):

ويعتمد التواصل بين العملاء على ثلاث اعمدة:

- لغة تواصل(langage de communication).
- بروتوكول الاتصال (protocole de communication) الذي يحدد كيفية الاتصال.
- دلالات الاتصال (sémantique de la communication) ونعني بها معنى الذي تتضمنه الرسالة المرسلة من طرف العميل.

البرمجة على منصة jade

La programmation sur la plateforme JADE

أهلا بك مجددا، انت الآن تمتلك فكرة جيدة عن الأنظمة متعددة العملاء، وحان الوقت لتبدأ ببرمجة الأفكار التي تحصلت عليها من الفصل الأول.

أقدم لك الفصل الموالي كنقطة بداية لك لبرمجة أنظمة متعددة العملاء على المنصة JADE مفترضا أنك لك خبرة كافية بلغة الجافا التي هي لغة البرمجة على هذه المنصة.

إن هذا الفصل يعد نقطة بداية وليس النهاية، حيث أنه يُحوصِل النقاط الأساسية لتبدأ بها طريقك نحو احتراف البرمجة في هذا المجال.

1. لمحة عن المنصة JADE:

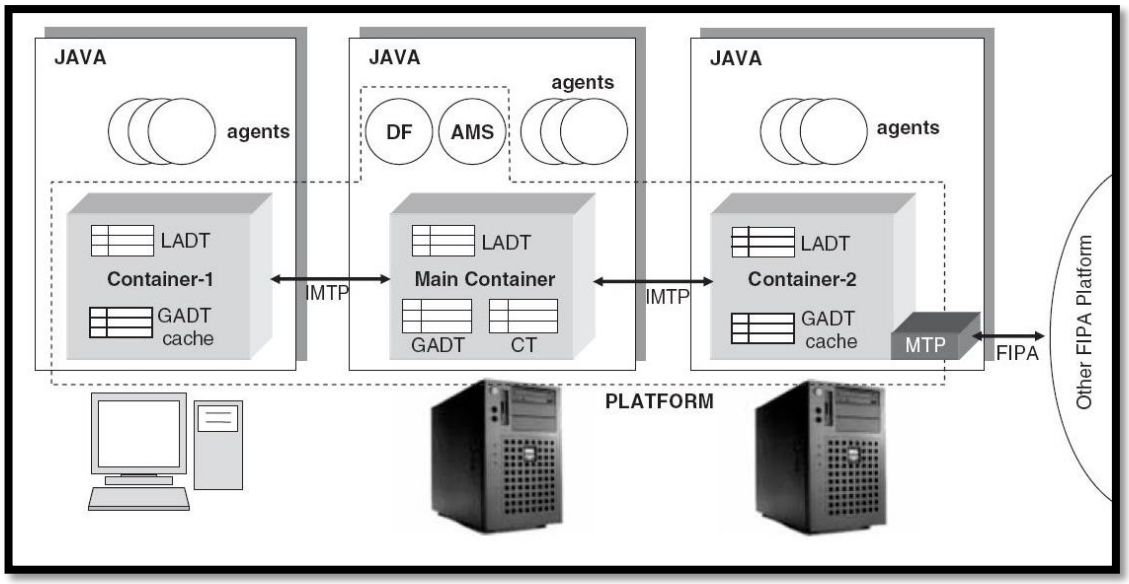
Jade هي اختصار لـ Java Agent DEvelopment Framework، وهي منصة برمجية مبرمجة ومنفذة بالكامل بواسطة لغة البرمجة جافا(Java) بهدف تسهيل تنفيذ الأنظمة متعددة العملاء من خلال طبقة برمجية وسيطة(middleware) متوافقة مع مواصفات FIPA[1]، ومجموعة من الأدوات الرسومية.

المنصة Jade لها موقع رسمي هو: http://jade.tilab.com، وفيه الملفات الاساسية للمنصة، دليل استخدامها، أمثلة، والمعلومات الاساسية لكفية العمل عليها.

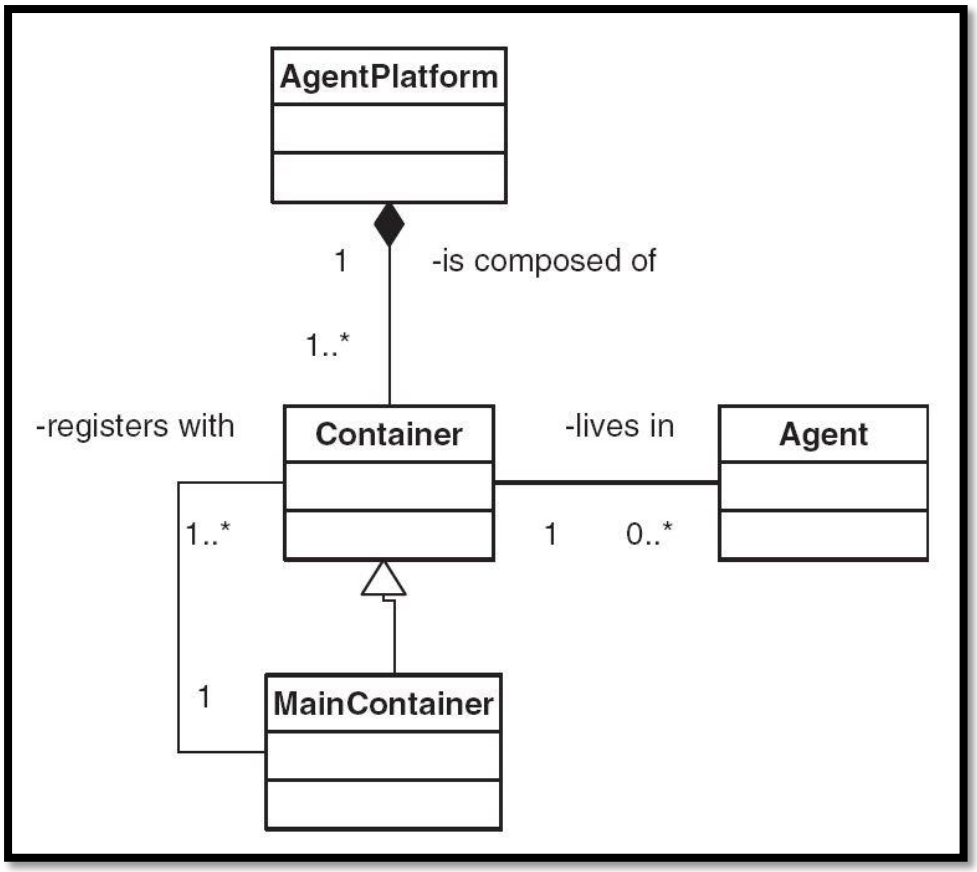
2. هندسة Jade (L'architecture du Jade):

المنصة Jade تتركب من agent containers (عبارة عن حاوية يعيش فيها العملاء) يمكن مشاركتها على الشبكة، حيث يعيش فيها العملاء، والمنصة توفر كل الخدمات التي نحتاجها لتنفيذ وادارة العملاء. هناك container خاصة تسمى: main container، وهي بمثابه النقطة الجامعة او الصندوق الجامع للمنصة، هي اول container يتم انشاؤها وكل باقي الـ containers يتم ربطهم مع الـ main container عن طريق التسجيل فيها، المخطط الموالي يوضح العناصر الاساسية للمنصة Jade.

[1] The Foundation for Intelligent Physical Agents(www.fipa.org)



مخطط الـ UML التالي يبين ويوضح العلاقات العناصر الاساسية للمنصة Jade.



افتراضيا تعطى الـ main container الاسم: 'Main Container' في حين تسمى البقية بـ 'Container-1', 'Container-2'... الخ.

لـ main container المسؤوليات الخاصة التالية:

- تسيير جدول الحاويات CT (the container table) والذي هو سجل معرفات وعناوين كل عناقيد الcontainers التي تتركب منها للمنصة.
- تسيير الجدول الوصفي العام للعملاء GADT (the global agent descriptor table) وهو سجل يضم جميع العملاء الذين تتضمنهم المنصة، ويحوي حالة كل عميل وموقعه.
- تسيير العميلين الخاصيين: AMS وDF .

3. تثبيت وتشغيل المنصة:

أ- تحميل الملفات:

كل الملفات المتعلقة بالمنصة تجدها على الموقع الرسمي للمنصة[1]، وتنقسم هذه الملفات الى قسمين: المنشورات الاساسية والاضافات. أما الاضافات فهي في الغالب ليست من انجاز فريق Jade ، بل من عمل مشتركي المصدر المفتوح. أما القسم الاساسي فيتكون من خمسة ملفات مضغوطة وهي:

- jadeBin.zip: ويضم ملفات المنصة في شكل Java archive (jar).
- jadeDoc.zip: يضم دليل المبرمج وكفية استخدام المنصة .
- jadeExamples.zip: يضم امثلة مختلفة مفتوحة المصدر.
- jadeSrc.zip: يحوي جميع اكواد المنصة Jade.
- jadeAll.zip: ويضم الملفات الاربعة السابقة مجتمعة.

[1] http://jade.tilab.com

بعد تحميل الملفات وفك الضغط عنها يتولد لدينا دليل يالشكل التالي:

```
jade/
    |---License
    |---classes
    |---demo
    |---doc/
    |      |---index.html
    |---lib/
    |      |---http.jar
    |      |---iiop.jar
    |      |---jade.jar
    |      |---jadeTools.jar
    |      |---commons-codec/
    |      |                |---commons-codec-1.3.jar
    |---src/
    |      |---demo
    |      |---examples
    |      |---FIPA
    |      |---jade
```

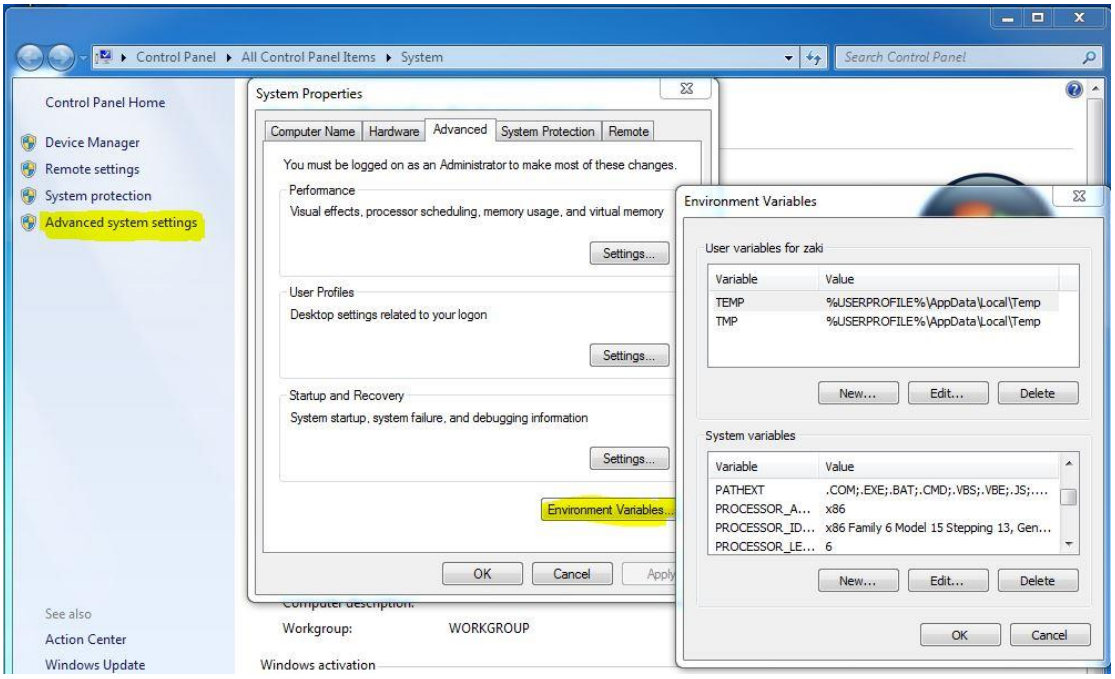
ملاحظات هامة:

- ✓ الملف jade/doc/index.html هو بمثابة مرجع يمكن الرجوع اليه، حيث يحوي روابط جيدة وهامة.
- ✓ الدليل jade/lib تتغير الملفات التي يحتويها حسب النسخة التي تم تحميلها، فمثلا النسخة JADE 4.3.1 (وهي اخر نسخة حتى كتابة هاته الاسطر وقد صدرت بتاريخ 2013/12/06) على غير نسخ اخرى سبقتها تحتوي على ملفين فقط في حين نجد ان نسخا اقدم تحوي اربعة ملفات كما توضحه الصورة السابقة، وهذ الملفات يجب ان يتضمنها الـ Java CLASSPATH في النظام.

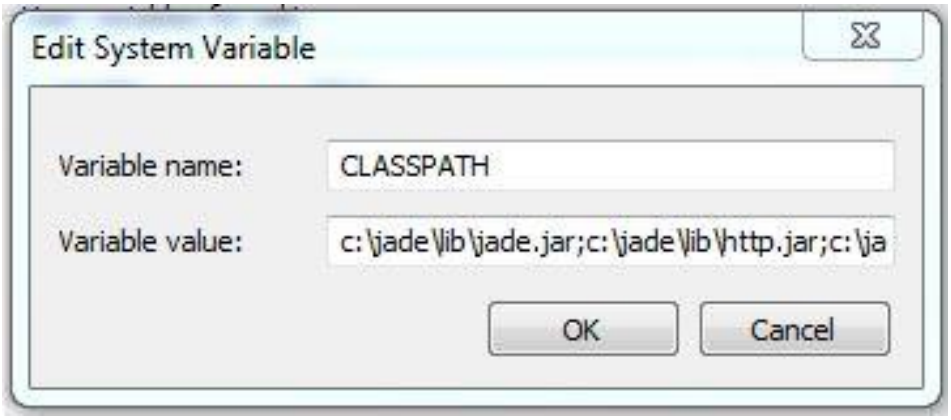
ب- تشغيل المنصة:

بعد تنزيل الملفات وفك الضغط عنها نقوم بتضمينها في الJava CLASSPATH الخاص بالنظام وتشغيل المنصة والطريقة كالتالي:

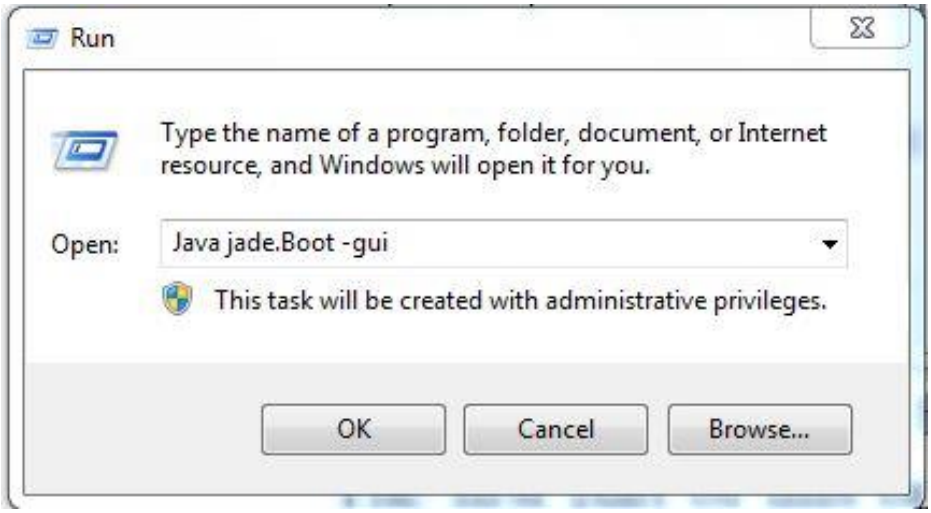
1- من نافذة خصائص النظام نضغط على خيارات متقدمة ثم تغيير متغيرات النظام.



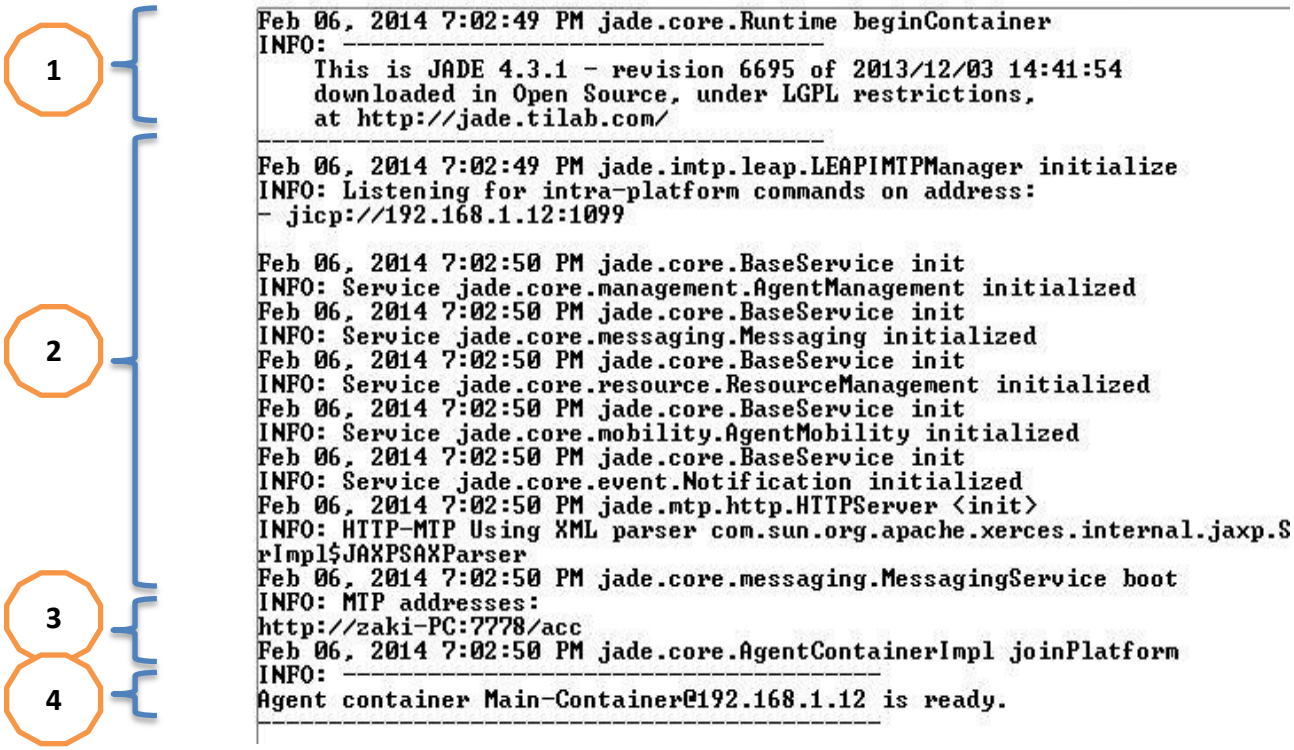
2- اذا كان المتغير CLASSPATH موجد نختاره ونضغط تعديل، أما اذا لم يكن موجود ننشئه بالضغط على جديد، نعطيه الاسم CLASSPATH والقيمة هي مسار الملفات الخاصة بالمنصة، مثلا: c:\jade\lib\jade.jar ، وفي حالة يوجد اكثر من ملف نضع مسارات جميع الملفات بحيث نفصل بين كل مسار واخر بفاصلة منقوطة.



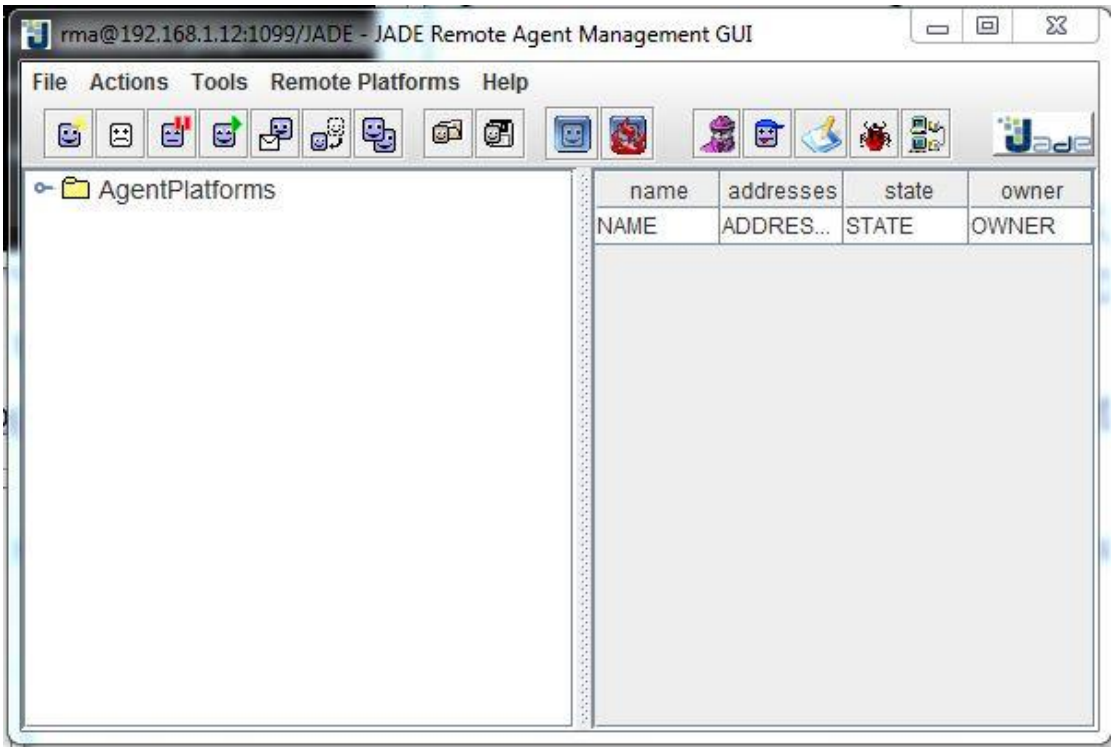
3- الان نأتي لتشغيل المنصة وذلك بتطبيق الامر java jade.Boot -gui اما عن طريق الدوس او نافذة تشغيل الموجودة بقائمة ابدا.



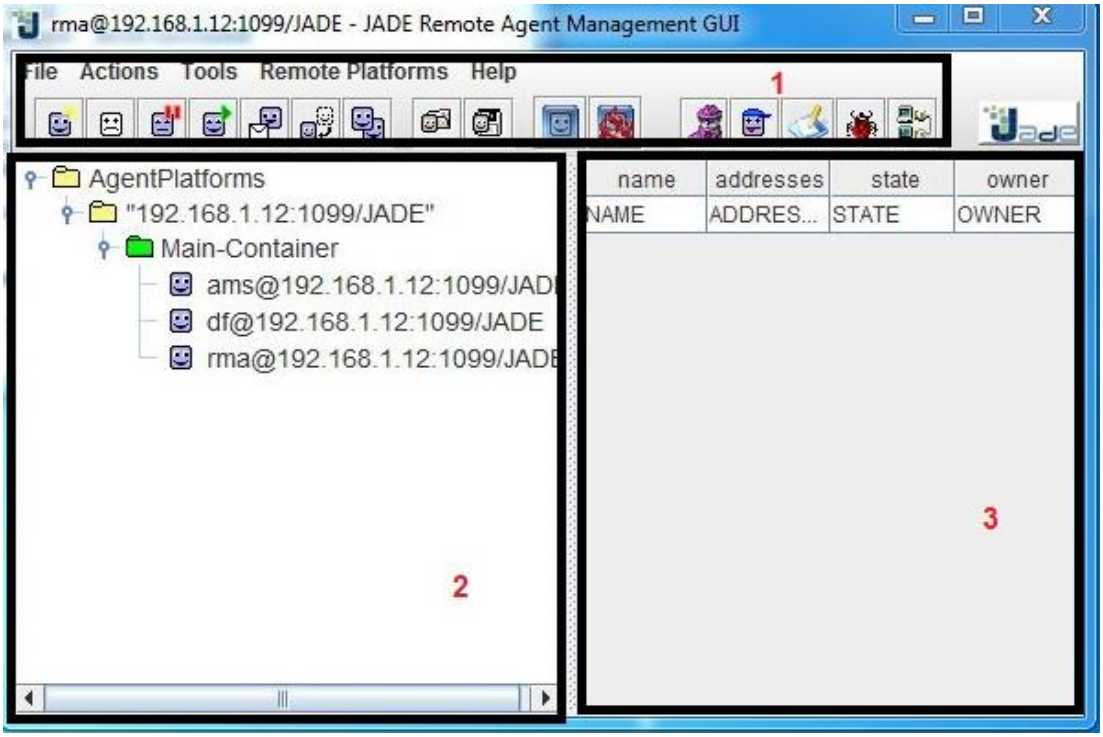
ستظهر نافذة شبيهة بنافذة الدوس تعرض لنا معض المعلومات، وتنقسم الى اربعة أجزاء، الجزء الاول يعرض معلومات عن نسخة المنصة وتوقيت اصدارها وقيود استخدامها. الجزء الثاني يعرض الخدمات القاعدية للمنصة والتي تم بالفعل انشاءها وجاهزيتها للاستخدام. الجزء الثالث يعرض عنوان المنصة الحالي، حيث يعرض هذا الجزء نوعين من العناوين MTP, HTTP ، أما الجزء الاخير فيعرض اسم الmain container جاهز والمنصة جاهزة للإستخدام.



وبعدها تظهر الواجهة الرسومية للمنصة وذلك نكون قد شغلنا المنصة على الجهاز.



ويمكن تقسيم الواجهة الى ثلاث اقسام رئيسية كما هو موضح بالصورة التالية:



القسم الأول يضم مجموعة الأوامر المتاحة، والثاني مخصص لعرض محتوى المنصة من الـ containers والعملاء في شكل شجري، أما القسم الثالث فيعرض معلوما العميل المحدد في القسم الثاني.

أخيرا لغلق المنصة نختار الامر الأخير من القائمة File.

4. العملاء الأساسيين على للمنصة:

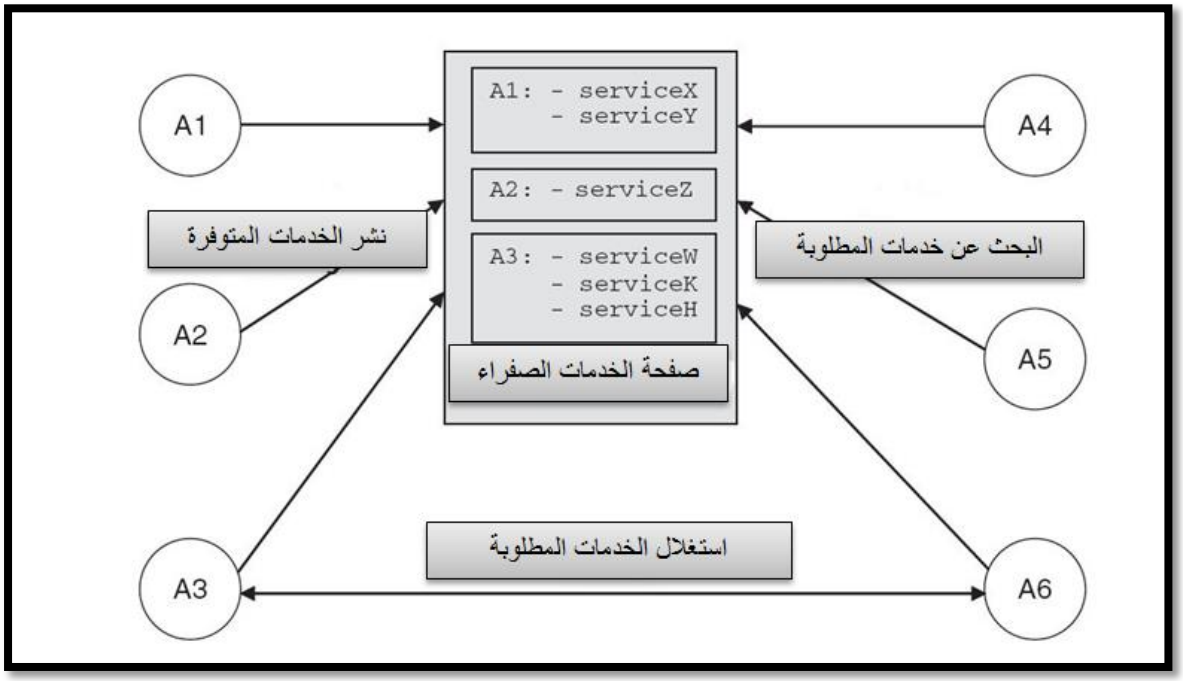
أ- العميل AP(Agent Platform):

وهو العميل الذي يوفر البنية الأساسية التي يعيش فيها العملاء. ويضم الآلات، نظام التشغيل، عناصر تسيير الخاصة بـFIPA، العملاء انفسهم وكل الإضافات.

ب- العميل DF (Directory Facilitator):

العميل DF (اسمه: df@<platform-name>) هو عميل خاص يقوم بتسيير الصفحة الصفراء(yellow pages)، وهي صفحة خاصة توفرها المنصة بغرض نشر وطلب الخدمات، حيث تسمح للعميل بوضع وصف للخدمات التي

يقدمها وهذا يسهل لباقي العملاء اكتشافها واستغلالها كما هو موضح بالمخطط التالي:



ت- العميل AMS (Agent Management System):

هذا العميل هو المسؤول عن اعطاء اسماء وعناوين العملاء حين انشائهم، حيث ان كل عميل يأخذ عنوانه من هذا العميل وهو نقطة الجمع بين كل العملاء الذين يريدون التواصل. العميل AMS هو العميل الوحيد المخول بتسيير العمليات على المنصة كإنشاء العملاء او انهائهم او انهاء container او انهاء عمل المنصة. العميل AMS بمثابة دليل لجميع العملاء المنتشرين على الAP، فهو يحوي معلوماتهم الاساسية من اسماء وعناوين وأيضا حالتهم الآنية(مفعل، متوقف....)[1]. وصف أي عميل يمكن تعديله بما يسمح به هذا العميل. عند حذف اي عميل فان هذه العميلة تنتهي بإلغاء تسجيله من العميل AMS وعند الغاء الAID الخاص بالعميل يصبح متاح لأي عميل يطلبه بعد ذلك. يمكننا الحصول على

[1] سنأتي على ذكر هذه الحالات في الحديث عن دورة حياة العميل.

وصف اي عميل عن طريق البحث داخل العميل AMS فهو القيم على ذلك، وذلك بطلب الاستعلام get-description.

5. الفئة Agent (عميل):

أ- كتابة وتنفيذ عميل:

لعمل عميل فإنه ما علينا سوى ان ننشئ فئة ترث من الفئة jade.core.Agent وبرمجة الدالة setup() الموروثة من هذه الفئة.

الدالة setup() يقصد بها تهيئة عناصر العميل بدلا من ان تتم هذه العملية في مُنْشِئ(constricteur) لأنه في حالة الانشاء لا يكون العميل مربوط بالمنصة وبالتالي لا تعمل بعض وظائفه بالشكل الصحيح.

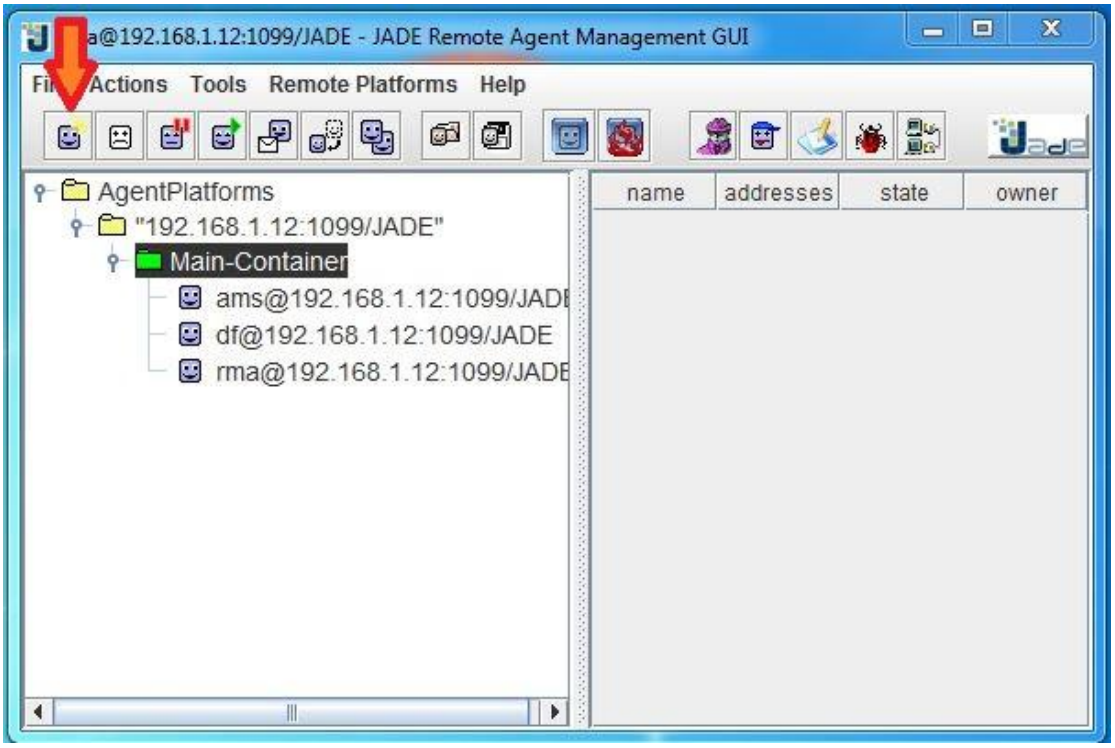
لنأخذ المثال hello المرفق مع ملفات المنصة وهو من أبسط الأمثلة ونص الكود به كالتالي(تم حذف التعليقات):

```
package examples.hello;
import jade.core.Agent;
public class HelloWorldAgent extends Agent {
  protected void setup() {
       System.out.println("Hello World! My name is "+getLocalName());
       doDelete();
  }
}
```

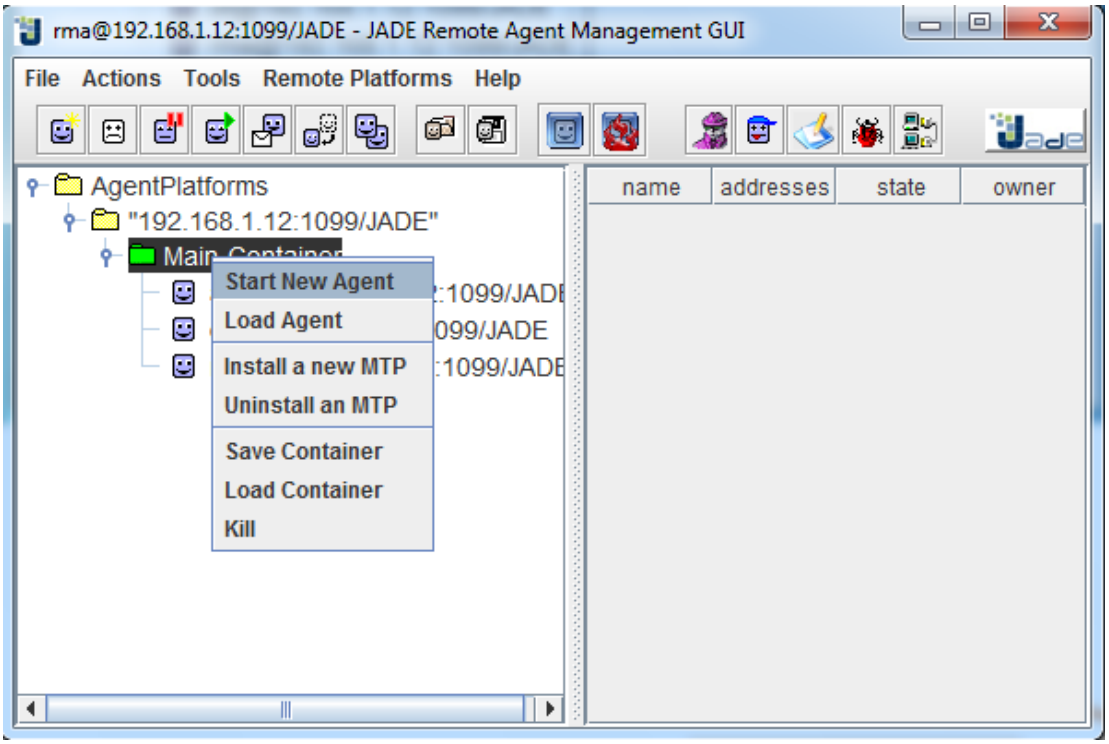
هذا المثال عبارة عن عميل يقوم بعرض رسالة ترحيبية مع ذكر اسمه عن طريق الدالة getLocalName() ومن ثم يموت عن طريق الدالة doDelete().

لتنفيذ هذا المثال باستعمال الواجهة الرسومية نحدد الcontainer التي نريد انشاء العميل فيها، ثم ننشئ العميل بإحدى الطرق الثلاثة التالية:

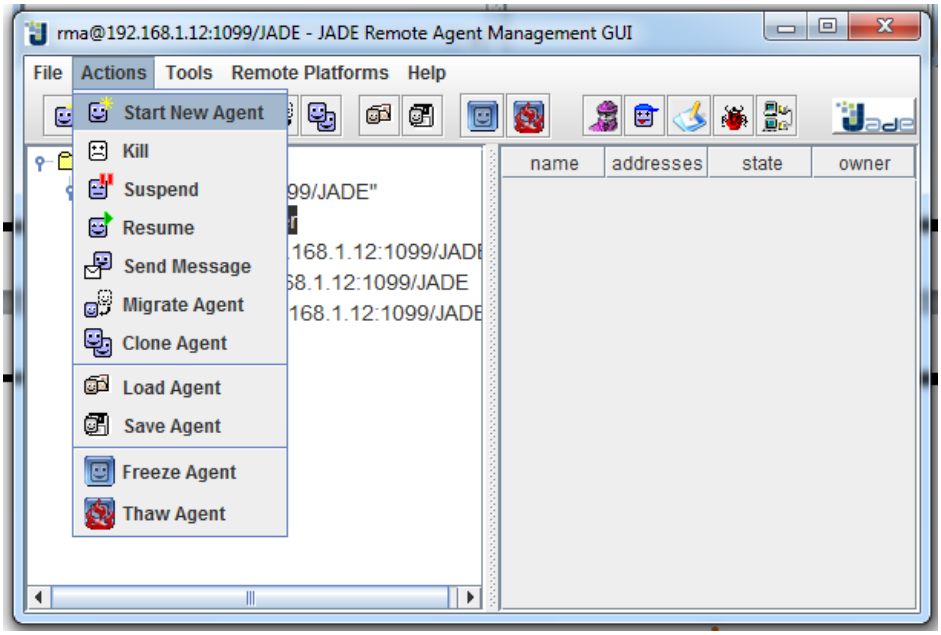
1- نضغط زر بدء عميل جديد المبين بالصورة



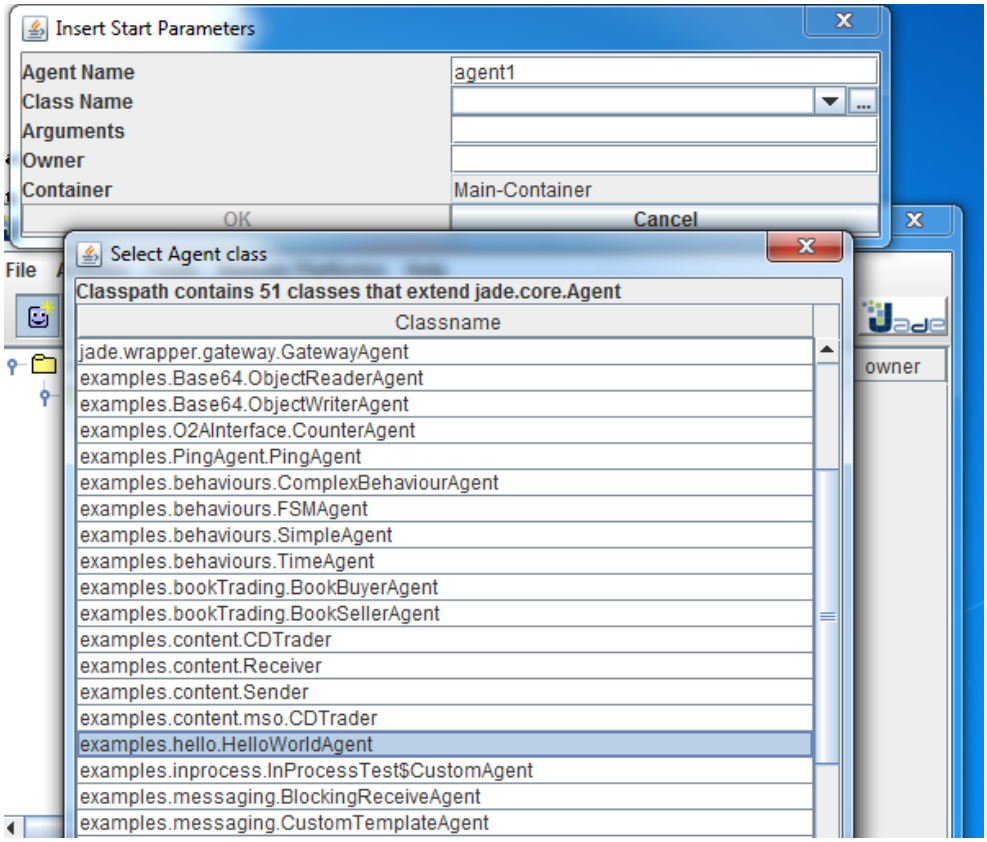
2- نضغط بالعكسية على الcontainer ونختار بدء عميل جديد



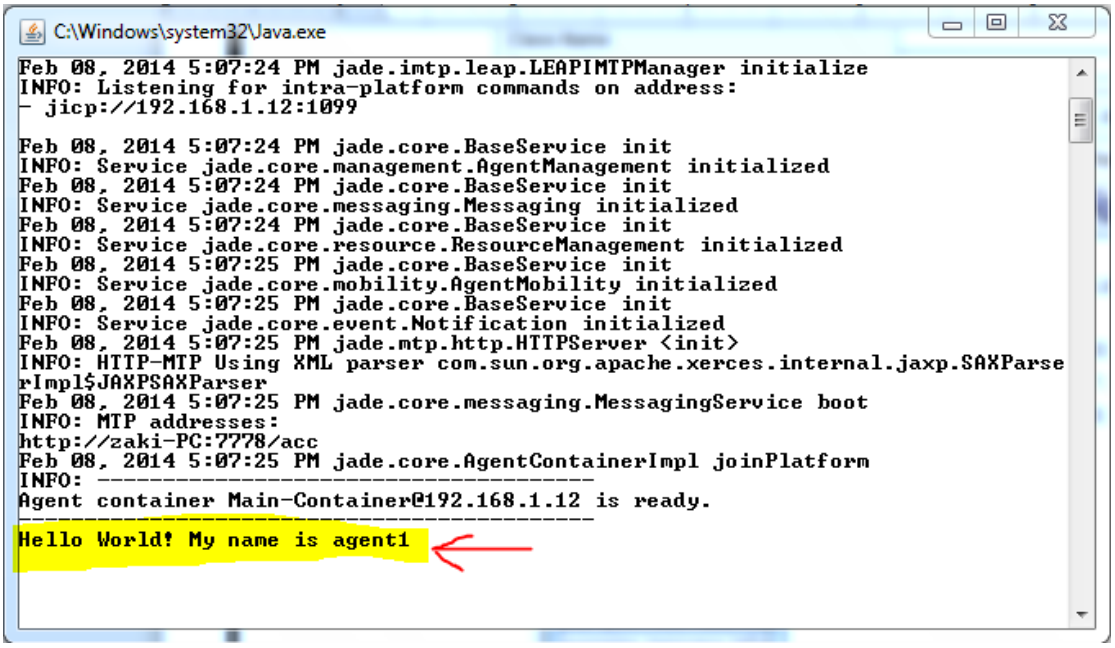
3- من القائمة Actions نختار الامر بدء عميل جديد



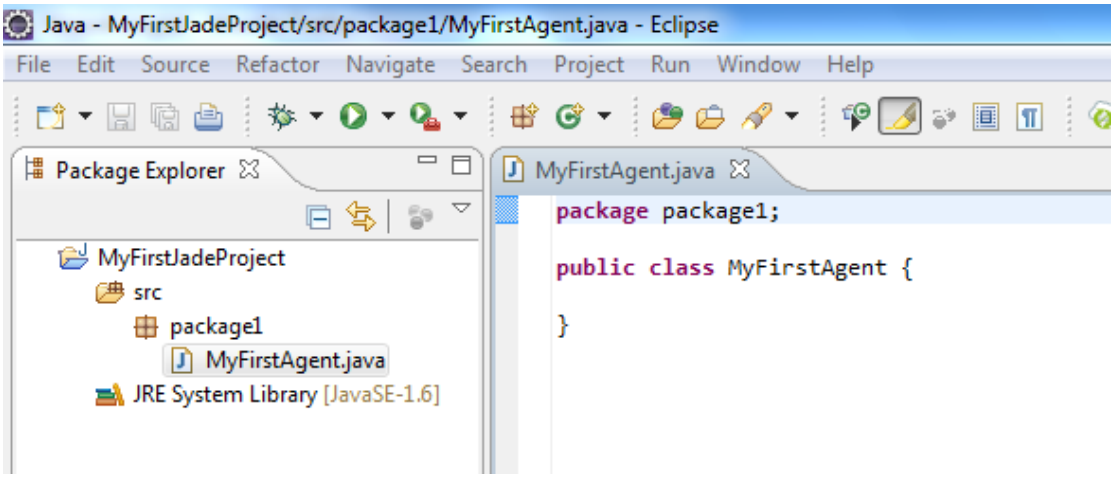
بتطبيق احد الطرق الثلاث ستفتح نافذة جديد لإعطاء المعلومات الاساسية للعميل كالاسم والفئة التي ينتمي اليها. في حالتنا أعطيناه الاسم agent1 والفئة examples.hello.HelloWorldAgent كما توضحه الصورة التالية:



بعد الضغط على موافق سيتم انشاء عميل جديد باسم agent1 يظهر رسالة كما هو موضح بالصورة لكن سيموت ولن نجده على المنصة.

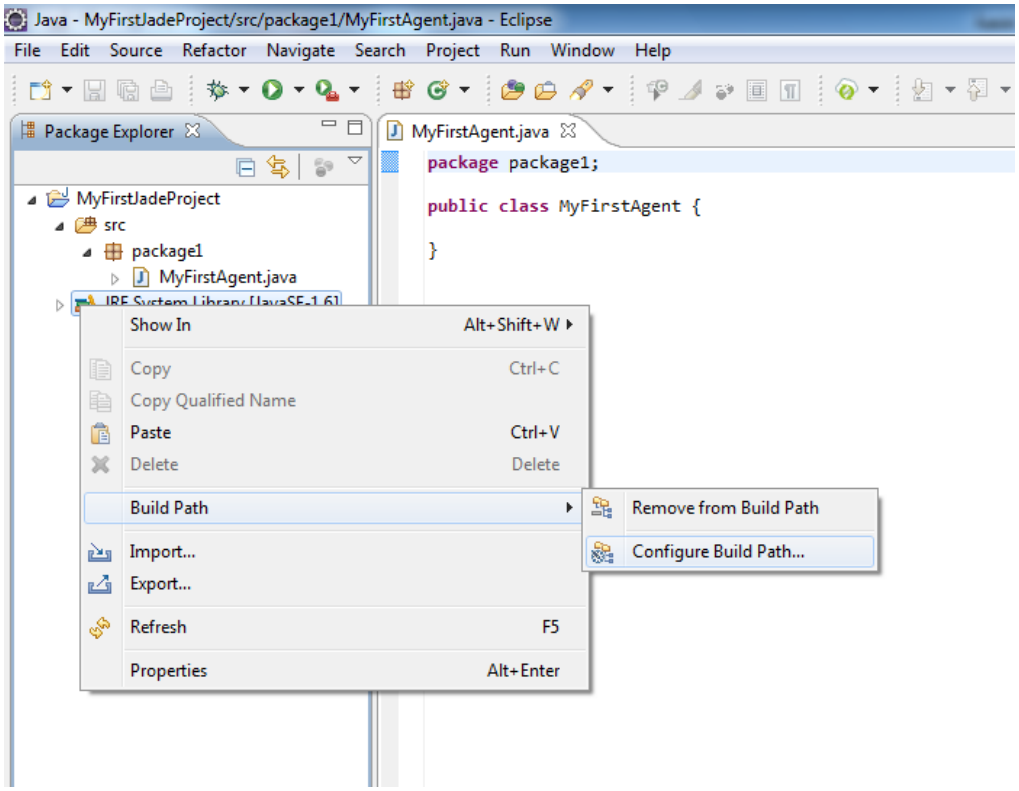


الأن لنقم بانشاء نفس العميل ولكن هذه المرة عن طريق الـ Eclipse. ولعمل ذلك ننشئ مشروع جديد على الـ eclipse باسم MyFirstJadeProject ثم ننشئ package باسم package1 ثم ننشئ داخلها فئة باسم MyFirstAgent فينتج لدينا الشكل التالي:

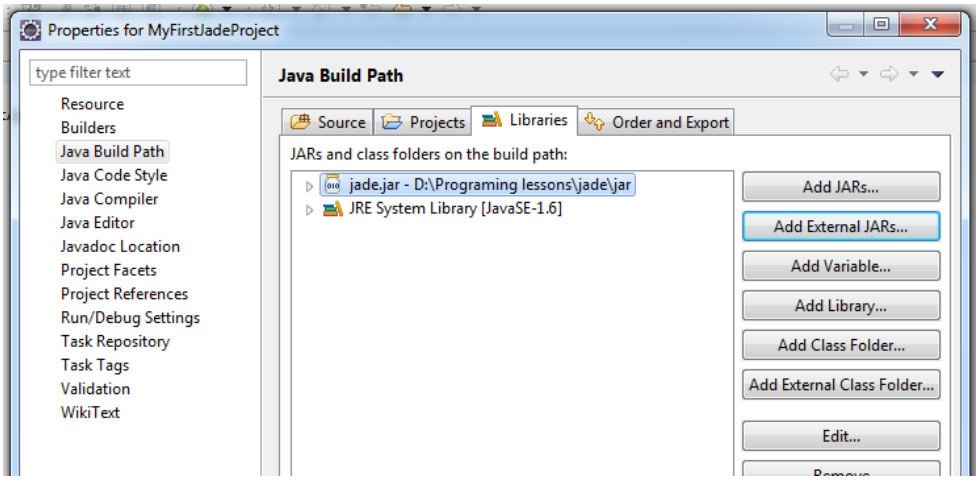


لكي نقول بان الفئة التي انشأناها هي عميل يجب ان ترث خصائص الفئة Agent كما وضحنا سابقا، ولعمل ذلك نتبع الخطوات التالية:

1- نُضمِّن المشروع كامل ملفات الـ jar الخاصة بالمنصة(كما اوردت من قبل فإن عدد الملفات متعلق بالنسخة التي يعمل عليها كل مبرمج، في هذا المثال اعمل على النسخة JADE 4.3.1، ويكفي ان احدد الملف jade.jar فقط) ولعمل ذلك تتبع الخطوات الموضحة بالصور التالية:



ستظهر نافذة لاضافة الملفات، نضغط الزر Add External JARs... ونحدد الملفات من على القرص.

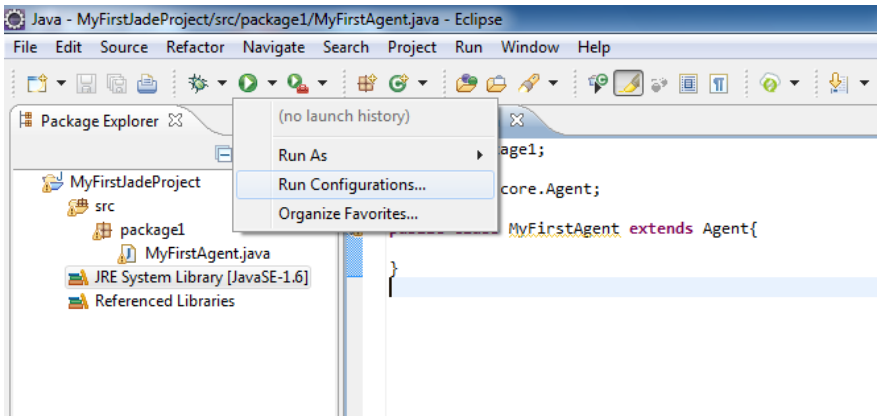


2- نعدل الكود الخاص بالفئة التي انشأناها ليصبح كالتالي(لاحظ أننا الان لم نستخدم اي دالة اضافية):

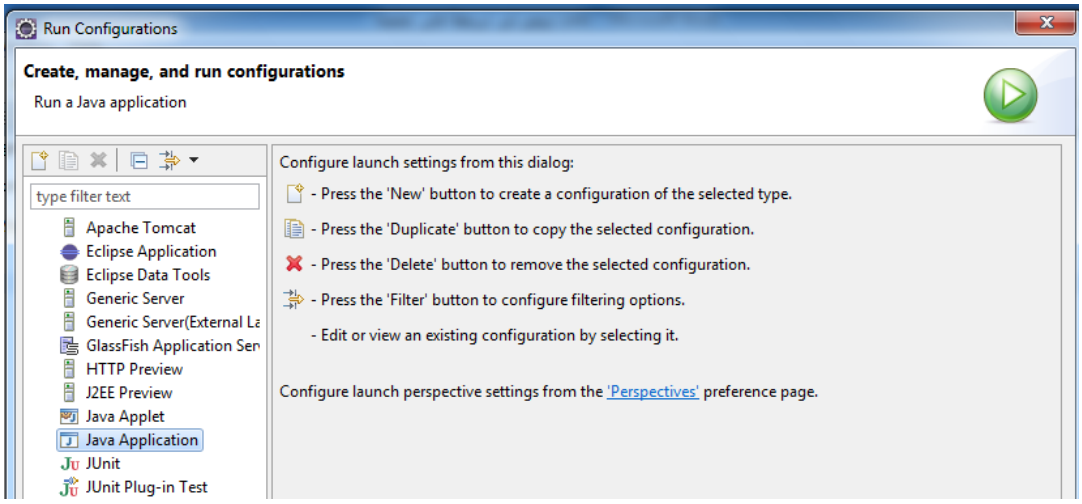
```
package package1;

import jade.core.Agent;

public class MyFirstAgent extends Agent{
	protected void setup() {
		// Printout a welcome message
		System.out.println("Hello World. I'm an agent!");
	}
}
```

الأن ننتقل للمرحلة الثانية وهي مرحلة التنفيذ التي تتطلب الخطوات التالية:

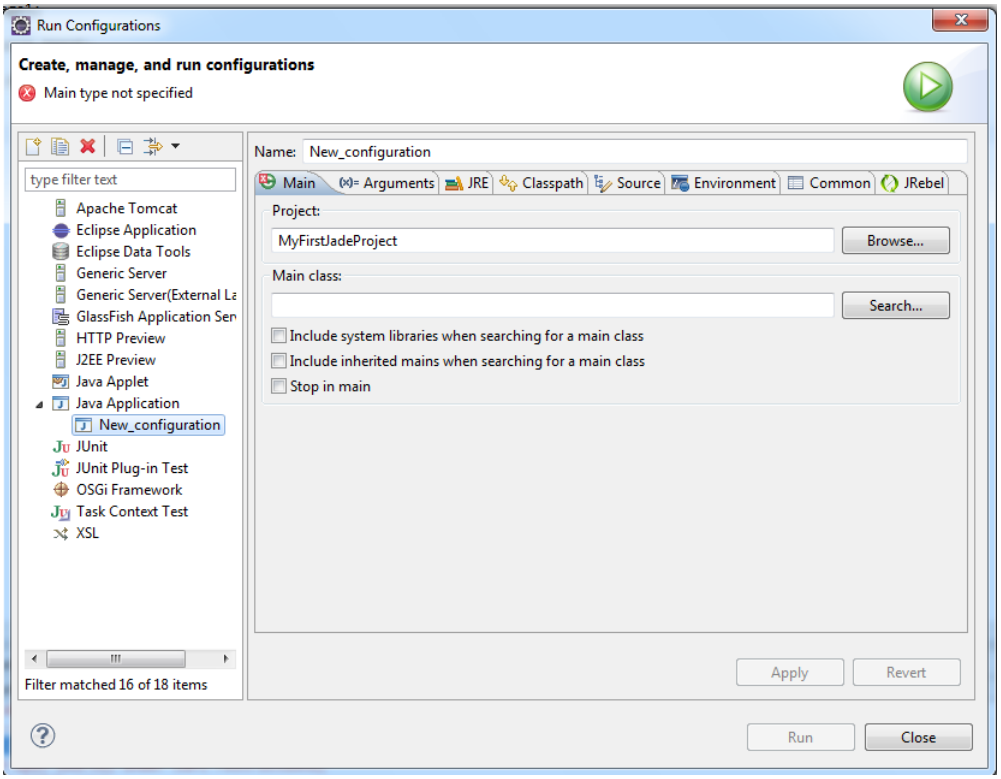
1- نفتح نافذة Run configuration، فهناك عدة طرق والصورة توضح احدى هاته الطرق



تظهر هذه النافذة



2- نضغط مرتين متتاليتين على Java Application لعمل اعدادات تشغيل جديدة



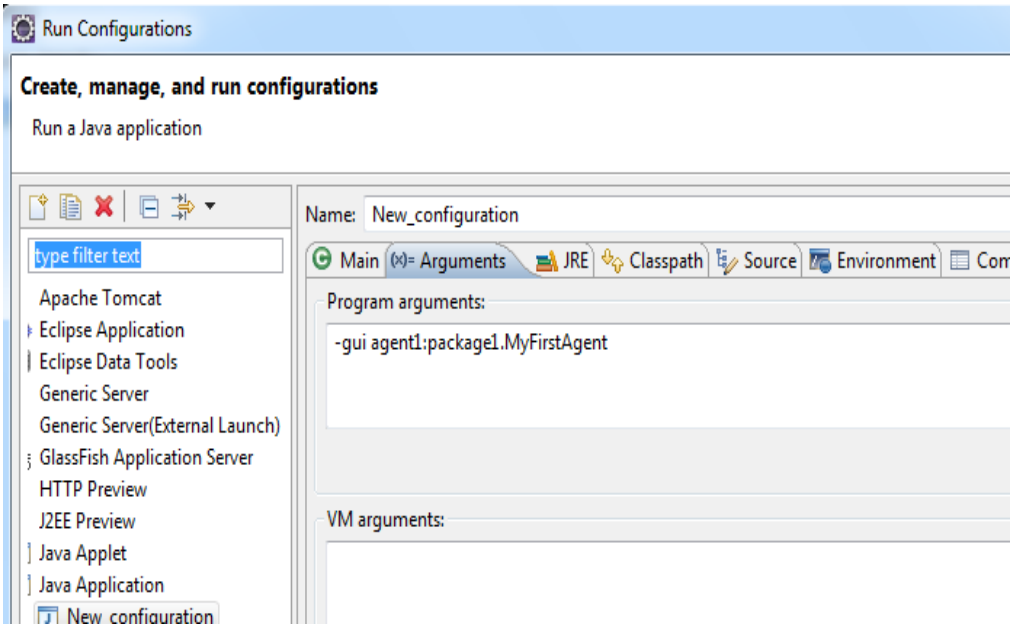
في الMain class نكتب jade.Boot (او نقوم بتحديده بالضغط على الزر Search...)، ثم نغير الى اللسان Arguments ونكتب في الـ Program arguments الكود: -gui agent1:package1.MyFirstAgent

وتقسيمه كالتالي:

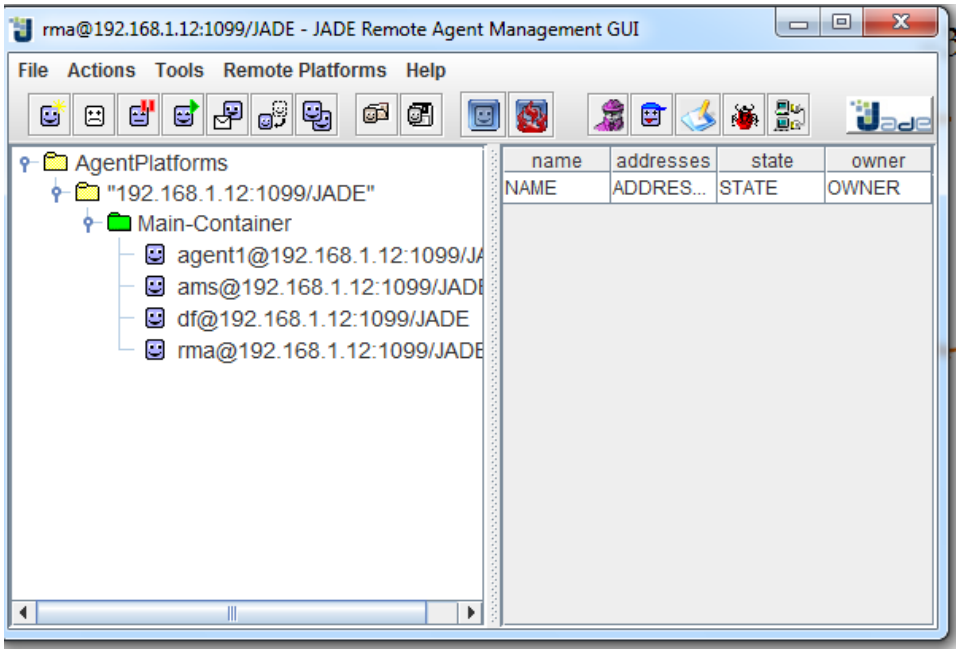
-gui : لعرض الواجهة الرسومية للمنصة.

agent1 : هو اسم العميل الجديد (طبعا يتغير من شخص الى اخر).

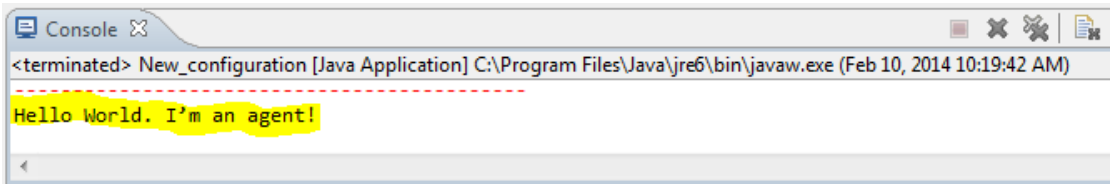
package1.MyFirstAgent : هو اسم الفئة



الان نضغط تطبيق(Apply) ثم تشغيل(Run)، ونشاهد عميل جديد تم انشاؤه باسم Agent1.



لاحظ أيضا انه تم عرض رسالة ترحيبية في النافذة Consol الخاصة بالبرنامج نصّها: `Hello World. I'm an agent!`.



لإنشاء اكثر من عميل نكرر الكود السابق بحيث لا نكرر امر تشغيل الواجهة الرسومية ومع مراعاة عدم تسمية أكثر من عميل بنفس الاسم مع الفصل بين الكود والثاني بفاصلة منقوطة.

```
-gui agent1:package1.MyFirstAgent;agent2:package1.MyFirstAgent
```

ب- إنشاء عميل عن طريق الكود:

الأكيد اننا نحتاج لإنشاء عميل عن طريق الكود، وليس من المنطق انه كلما اردنا انشاء عميل جديد ان نستخدم الواجهة او عن طريق تغيير اعدادات التشغيل، ومن جهة اخرى، لن ننشئ كل العملاء دفعة واحدة بل الحاجة تحتم علينا ان ننشئ العملاء عن طريق الكود كأن نعطي صلاحيات الانشاء الى عميل رئيسي يقوم بهذا الأمر في وقته عند الحاجة إليه او عن طريق برامج اخرى، فليس من المنطقي أيضا ان ننشي عميلا منذ بدء تشغيل البرنامج ونحن لا نحتاجه الا في آخر العمل.

إن للمنصة JADE وقت التنفيذ (run-time) والمنفذة من قبل الفئة jade.core.Runtime نسخة(instance) واحدة فقط. حيث أنه نسخة واحدة فقط من هذه الفئة يمكن ان تكون موجودة في JVM[1] ويمكن الوصول اليها عن طريق الدوال الثابتة (static method). وهي تتوفر على الدالتين createMainContainer() رئيسية والدالة container لإنشاء createAgentContainer() فرعية (والتي تكون container لإنشاء مرتبطة مع اخرى رئيسية). كلا الدالتين في حاجة الى الكائن Profile كمُدْخَلات(parameter) لضبط خيارات تهيئة وبدء تنفيذ المنصة JADE (اسم الخادم، رقم المنفذ، الmain container).

[1] Java virtual machine

جميع الخيارات متاحة كثوابت(constants) في الفئة Profile ويمكن التعديل عليها عن طريق الدالة setParameter(String key, String value).

كل من الدالتين createMainContainer() و createAgentContainer() ترجعان كائن من الفئة jade.wrapper.ContainerController. هذا الكائن يضم عدة وضائف عالية المستوى مثل تثبيت او إلغاء تثبيت MTPs، الغاء container وأيضا انشاء عميل جديد.

الدالة createNewAgent() من الفئة ContainerController تأخذ كمُدخلات(parameter) ثلاث متغيرات، الأول هو سلسلة حرفية يمثل اسم العميل الجديد، والثاني سلسلة حرفية وهو الفئة التي ينتمي اليها العميل، أما الأخير فيتضمن مجوعة الarguments، وترجع كائن من نوع AgentController والذي بدوره يهتم بعد وظائف العميل مع الحفاظ على استقلاليته. على وجه الخصوص فأن الAgentController يوفر دوال تتيح للبرامج الخارجية ان تتحكم في دورة حياة العميل مع اخفاء الرابط(reference) اليه، لأنه ليس ممكنا تنفيذ دوال عليه بشكل مباشر. لاحظ ان الدال createNewAgent() تنشئ العميل فقط ولا يبدأ في عمله، لأن هذا لا يمكن تحقيقه إلا من خلال الدالة start() للكائن المُرجَع من قبل الAgentController.

الكود التالي يعطينا مثال عن انشاء عميل عن طريق برنامج خارجي:

```
// Get a hold on JADE runtime
Runtime rt = Runtime.instance();
// Create a profile
Profile p = new ProfileImpl();
// Create a new non-main container, connecting to the default
// main container (i.e. on this host, port 1099)
ContainerController cc = rt.createAgentContainer(p);
// Create a new agent and start it
AgentController ac;
try {
	ac = cc.createNewAgent("agent-name","agent-class", null);
	ac.start();

} catch (StaleProxyException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

يمكننا تعديل خصائص الProfile كما سبق الاشارة اليه كالتالي:

```
p.setParameter(Profile.MAIN_HOST, "host-name");
p.setParameter(Profile.MAIN_PORT, "port-number");
```

ويمكننا أيضا استخدام الدالة acceptNewAgent() بدل الدالة createNewAgent() وذلك باعطائها مُستنسخ من العميل والاسم الذي سيحمله فقط

```
try {
    ac = cc.acceptNewAgent("agent-name ", new Agent());
    ac.start();
} catch (StaleProxyException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

واذا اردنا ان ينشيء عميل ما عميل اخر بنفس الcontainer فإننا نستخدم نفس الContainerController الخاص به كالتالي:

```
protected void setup() {
	ContainerController cc;
	AgentController ac;
	cc= getContainerController();
	try {
		ac = cc.acceptNewAgent("agent", new Agent());
		ac.start();
	} catch (StaleProxyException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
```

ويمكن اختصار نفس الكود في سطر واحد كالتالي:

```
try {
	getContainerController().acceptNewAgent("agent", new Agent()).start();
} catch (StaleProxyException e1) {
	// TODO Auto-generated catch block
	e1.printStackTrace();
}
```

ت- معرفات العميل:

يتم تعريف عن طريق معرف العميل(agent identifier)، على المنصة يكون معرف العميل عبارة عن كائن مُستنسخ من الفئة jade.core.AID ، ويحوي اسم العميل العام(GUID[1]) والاسم المحلي على المنصة التي يعيش عليها وأيضا العناوين الخاصة بالعميل والتي من خلالها يمكن التواصل معه من قبل باقي العملاء. الدالة getAID() في العميل تسمح لنا بجلب معرف العميل وبالتالي الوصول الى كل معلومات الاتصال به. الاسم في المنصة Jade يكون على الشكل التالي:

<اسم المنصة>@<اسم العميل>

مثال: Agent1@zaki-platform

الفئة AID توفر لنا دوال تسمح لنا باسترجاع الاسم المحلي للعميل (getLocalName())، الاسم العام (getName())، والعنوانين الخاصة بالعميل(getAllAddresses())

جرب الدوال السابقة بتعديل الدالة setup() الخاصة بالمثال السابق لتصبح كما يلي:

[1] globally unique name

```
protected void setup() {
	// Printout a welcome message
	System.out.println("Hello World. I'm an agent!");
	System.out.println("My local-name is" + getAID().getLocalName());
	System.out.println("My GUID is "+getAID().getName());
	System.out.println("My addresses are:");
	Iterator it = getAID().getAllAddresses();
	while (it.hasNext()) {
		System.out.println("- "+it.next());
	}
}
```

الاسم المحلي العميل الواحد لا يمكن ان يسند لأكثر من عميل على نفس المنصة، ويمكننا ان ننشئ معرف عميل بالشكل التالي:

```
String localname = "Agent_name";
AID id = new AID(localname, AID.ISLOCALNAME);
```

ونفس الأمر بالشكل التالي:

```
String guid = "Agent_namt@platform_name";
AID id = new AID(guid, AID.ISGUID);
```

ث- دورة حياة العميل:

العميل على المنصة Jade يكون في حالة من الحالات التالية ينتقل بينها حسب الحاجة:

- **يُهيّأ(Initiated)**: في هذه الحالة يكون العميل في إطار الانشاء لكنه لم يتم تسجيله مع العميل AMS، ليس له اسم ولا عنوان ولا يمكنه التواصل مع باقي العملاء.
- **مفعل(Active)**: في هذه الحالة يكون العميل قد تم تسجيله وحصل على اسم وعنوانه ويمكنه التواصل مع باقي العملاء ويمكنه استغلال الوظائف التي تقدمها المنصة.

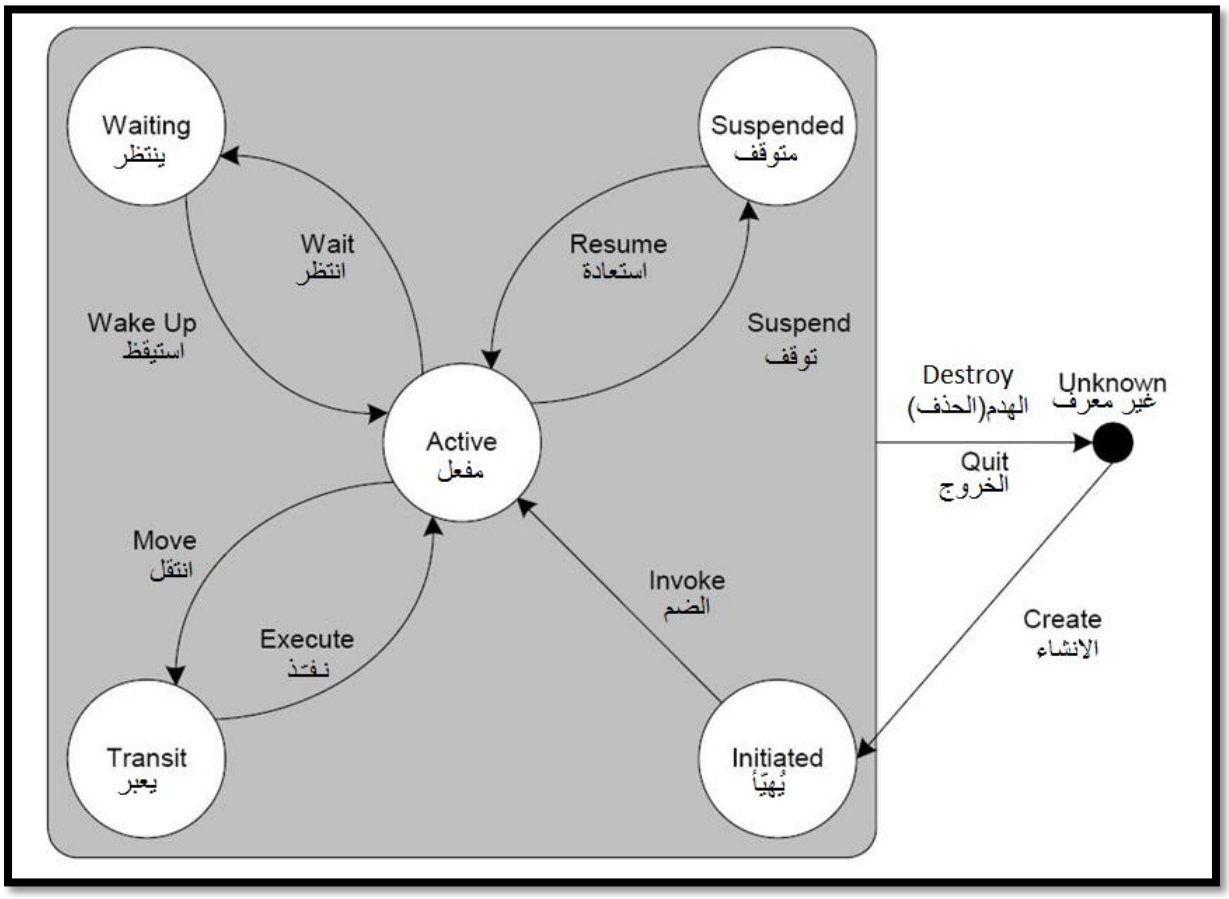
❖ **موقف(Suspended)**: العميل حاليا متوقف. كل برامجه الداخلية(thread) متوقفة ولا يوجد اي مهمة(behaviour) في اطار التنفيذ.

❖ **ينتظر(Waiting)**: العميل متوقف، ينتظر شيء ما. برامجه الداخلية(thread) في حالة نوم وستستيقظ في حال توفرت شروط معينة(عادة يستيقظ عند ورود رسالة).

❖ **محذوف(Deleted)**: يكون العمل قد مات. برامجه الداخلية تمّ تنفيذها والعميل لم يعد مسجلا لدى العميل AMS.

❖ **العبور()**: العميل المتنقل يكون في هذه الحالة عند هجرته من موقع الى لآخر.

المخطط التالي يوضح انتقال العميل من حالة الى اخرى:



الفئة Agent توفر مجموعة من الدوال التي تسمح بإنتقال العميل من حالة الى اخرى، مثلا الدالة doWait() تجعل العميل نتقل الى حالة الانتظار، والدالة doSuspend() تجعل العميل يتحول الى حالة التوقف وهكذا، يمكنك الرجوع الى المراجع والروابط التي توفرها المنصة والتي تخص الفئة Agent للحصول على القائمة الكاملة للدوال التي من الشكل: doXXXX()[1].

ملاحظة هامة: العميل لا يستطيع ان ينفذ المهمات(behaviours) الخاصة به الا في الحالة التي يكون فيها مفعل، واذا ما تم استدعاء الدالة doWait() فان العميل وكل مهام العميل تتوقف وليس فقط المهمة الذي استدعيت فيها هذه الدالة. الى جانب ذلك فان الدالة block() هي جزء من الفئة Behaviour لأجل ان تسمح بإنهاء مهمة معينة.

6. الفئة Behaviour (مهمة):

ان الفئة Behaviour تضم كل انواع المهام التي يمكن ان يقوم بها العميل، فهي تنفذ في شكل كائن من الفئة jade.core.behaviours.Behaviour(لاحظ أن الترجمة الحرفية للكلمة Behaviour هي سلوك وهي الترجمة المرادفة لـ Comportement بالفرنسية، لكن ارى الترجمة الاقرب من حيث المضمون هي "مهمّة" وجمعها "مهام").

لجعل العميل ينفذ مهمة جديدة عن طريق كائن(object)، نستعمل الدالة addBehaviour(Behaviour) الخاصة بالفئة Agent ليتم إضافتها الى قائمة المهام التي يجب ان ينفذها العميل، ويمكننا استدعاء هاته الدالة في اي وقت نحتاجها بعد بداية عمل العميل(في الدالة setup()). من جهة أخرى لالغاء مهمة ما من قائمة مهام

[1] http://jade.cselt.it/doc/api/jade/core/Agent.html

العميل نستدعي الدالة removeBehaviour(Behaviour) والتي تتكفل بهذه العملية.

عند العمل على الفئة Behaviour فانه يتوجب علينا برمجة فئتين مجردتين(abstract) فيها، الدالة action() ويتم فيها تنفيذ العمليات الخاصة بالمهمة المراد تنفيذها، الدالة done() وهي دالة ترجع قيمة منطيقية(boolean)، ويبدأ تنفيذها بعد ارجاع الدالة action() وفيها يتم تحديد ما اذا كانت المهمة انتهت او لا.

الدالة block() تنهي المهمة كأنما ان الدالة قد أرجعت(المقصود بأرجعت أن الدالة قد انتهت وارجعت قيمة). الى جانب الدالتين يوجد دالتين هممتين ولكن برمجتهما ليس مشروط وهما: الدالة onStart() ويتم استدعاؤها قبل الدالة action() مباشرة، والثانية هي الدالة onEnd() ويتم استدعاؤها مباشرة بعد ان ترجع الدالة done() القيمة true.

في العديد من الاحيان نحتاج الى معرفة العميل المنفذ للمهمة والوصول الى خصائصه من دوال ومتغيرات، فمثلا اذا كانت المهمة تنتهي بقتل العميل فلابد ان يوجد رابط بين الـ Behaviour والعميل، ويتم ذلك عن طريق المتغير myAgent، مثلا لانهاء العميل ننفذ الامر myAgent.doDelete().

يوجد عدة اشكال من المهمات التي ينفذها العميل بناءً على طريقة تنفيذه لها، ويمكننا تقسيمها الى ثلاث أقسام رئيسية:

أ-مهمات بسيطة:

❖مهمة واحدة قصيرة(One-shot Behaviour):

مع هذا النوع تنفذ المهمة مرة واحدة ووحيدة. وهي عبارة عن كائنات مستنسخة من الفئة jade.core.behaviours.OneShotBehaviour وفيه الدالة done() مبرمجة مسبقا حيث ترجع دائما القيمةtrue .

```
public class MyOneShotBehaviour extends OneShotBehaviour {
	public void action() {
		// perform operation X
	}
}
```

العملية X تنفذ مرة واحدة.

❖مهمة تكرارية(Cyclic Behaviour):

كما هو واضح من اسمها فغن المهمة مع هذا النوع تنفذ بشكل تكراري.

هي عبارة مستنسخ من الفئة jade.core.behaviours.CyclicBehaviour

وفيه الدالة done() مبرمجة مسبقا حيث ترجع دائما القيمةfalse .

```
public class MyCyclicBehaviour extends CyclicBehaviour {
	public void action() {
		// perform operation Y
	}
}
```

العملية Y تنفذ بشكل متكرر.

❖مهمة عامة (Generic Behaviour):

الفرق بين هذا النوع وما سبق، أنه لم تتم برمجة الدالة done() وتُرك ذلك للمبرمج ليحدد كيف تنتهي المهمة على حسب حاجته، وهذا النوع من المهام هو عبارة كائنات مستنسخة من الفئة jade.core.behaviours.Behaviour

.

```
public class ThreeStepBehaviour extends Behaviour {
	public void action() {
		// perform operation Z
	}
	public boolean done() {
		return <codition>;
	}
}
```

❖ WakerBehaviour():

في هذا النوع تم برمجة الدالتين action() و done() مسبقا ليتم تنفيذ الدالة المجردة onWake() بعد مدة زمنية محددة بالملي ثانية يتم إدخالها من خلال المُنشئ(constructor). وهذا النوع من المهام هو عبارة كائنات مستنسخة من الفئة jade.core.behaviours.WakerBehaviour.

```
public class MyAgent extends Agent {
	protected void setup() {
		System.out.println("Adding waker behaviour");
		addBehaviour(new WakerBehaviour(this, 10000) {
			protected void onWake() {
				// perform operation X
			}
		} );
	}
}
```

في هذا المثال سيتم تنفيذ العملية X بعد 10 ثواني.

❖ TickerBehaviour():

في هذا النوع تم برمجة الدالتين action() و done() مسبقا ليتم تنفيذ الدالة المجردة onTick() بشكل متتابع بفارق مدة زمنية محددة بالملي ثانية يتم إدخالها من خلال المُنشئ(constructor).

وهذا النوع من المهام هو عبارة كائنات مستنسخة من الفئة jade.core.behaviours.TickerBehaviour.

```
public class MyAgent extends Agent {
	protected void setup() {
		addBehaviour(new TickerBehaviour(this, 10000) {
			protected void onTick() {
				// perform operation Y
			}
		} );
	}
}
```

العملية Y يتم تنفيذها مرة كل 10 ثواني.

ب- المهام المركبة(The composed Behaviours):

يضم هذا القسم ثلاث انواع من المهام المركبة من عدة مهام اخرى:

❖مهمات المتتابعة(Sequential Behaviour):

وهي عبارة عم مهمة واحدة مكونة من عدة مهام فرعية، حيث يتم تنفيذ المهام الفرعية بشكل متتالي. الدالةaddSubBehaviour() الخاصة بالفئة SequentialBehaviour تسمح بإضافة مهمة فرعية جديد، في حين الدالةremoveSubBehaviour() تسمح بحذف مهمة مضافة من قبل.وهذا النوع من المهام هو عبارة كائنات مستنسخة من الفئة jade.core.behaviours.SequentialBehaviour.

```
protected void setup() {
	SequentialBehaviour sequentialBehaviour =
		new  SequentialBehaviour(this);
	sequentialBehaviour.addSubBehaviour(new OneShotBehaviour(this) {
		public void action() {
			// perform operation X
		}
	} );
	sequentialBehaviour.addSubBehaviour(new OneShotBehaviour(this) {
		public void action() {
			// perform operation Y
		}
	} );
	sequentialBehaviour.addSubBehaviour(new OneShotBehaviour(this) {
		public void action() {
			// perform operation Z
		}
	} );
}
```

المهمات X،Y و Z تنفذ بشكل متتابع(Xثم Yثم Z).

❖مهمات المتوازية(Parallel Behaviour):

هذا النوع يسمح لنا بتنفيذ اكثر من مهمة في آن واحد. وهو عبارة عن كائنات مستنسخة من الفئة jade.core.behaviours.ParallelBehaviour.

لجعل هذا النوع ينهي جميع المهام مع نهاية اي مهمة فرعية نمرر العبارة WHEN_ANY في المُنشئ، واذا اردنا العكس، اي بعد ان تنفذ جميع المهام الفرعية نمرر العبارة WHEN_ALL.

```
protected void setup() {
	ParallelBehaviour parellelBehaviour =
		new ParallelBehaviour
			(this,ParallelBehaviour.WHEN_ANY);
	parellelBehaviour.addSubBehaviour(
			new OneShotBehaviour(this) {
		public void action() {
			// perform operation X
		}
	} );
	parellelBehaviour.addSubBehaviour(
			new OneShotBehaviour(this) {
		public void action() {
			// perform operation Y
		}
	} );
	parellelBehaviour.addSubBehaviour(
			new OneShotBehaviour(this) {
		public void action() {
			// perform operation Z
		}
	} );
}
```

المهمات X،Yو Z تنفذ معا في نفس الوقت، وحين ينتهي تنفيذ واحدة من هاته المهام تنتهي باقي المهام في نفس الوقت لاننا مررنا العبارة WHEN_ANY للمُنشئ.

❖مهمات معقدة(Composite Behaviour):

وتنفذ في شكل كائنات (opjects) مستنسخة من الفئة jade.core.behaviours.FSMBehaviour.

وهي نوع من finite state automaton(FSA) أو finite state machine(FSM) ، لا أعرف بالضبط الترجمة للعربية، وجدت من ترجمها بـ **آلة الحالات المحدودة** أما بالفرنسية: automate à états finis أو machine à états finis. وهي عبارة عن خريطة ترسم انتقال العميل من تنفيذ مهمة الى اخرى بناءٌ على النتائج التي تحصل عليها من تنفيذ آخر مهمة، ويطلق على كل مهمة بالحالة (state) وتعطى كل حالة اسم، و يوجد حالة ابتدائية واحدة تبدأ بها الخريطة في حين يمكن أن نجد أكثر من حالة يمكن أن ينتهي اليها العميل. ونسمي التحول من حالة الى اخرى بالعبور او الإنتقال(Transition). حيث:

لإضافة حالة جديدة نستخدم الدالة:

registerState(Behaviour state, String name)

لإضافة الحالة الابتدائية نستخدم الدالة:

registerFirstState(Behaviour state, String name)

لإضافة حالة نهائية نستخدم الدالة

registerLastState(Behaviour state, String name)

لإضافة عبور من حالة الى اخرى نستخدم الدالة

registerTransition(String s1, String s2, int event)

لإضافة عبور افتراضي نستخدم الدالة

registerDefaultTransition(String s1, String s2, String[] toBeReset)

حيث:

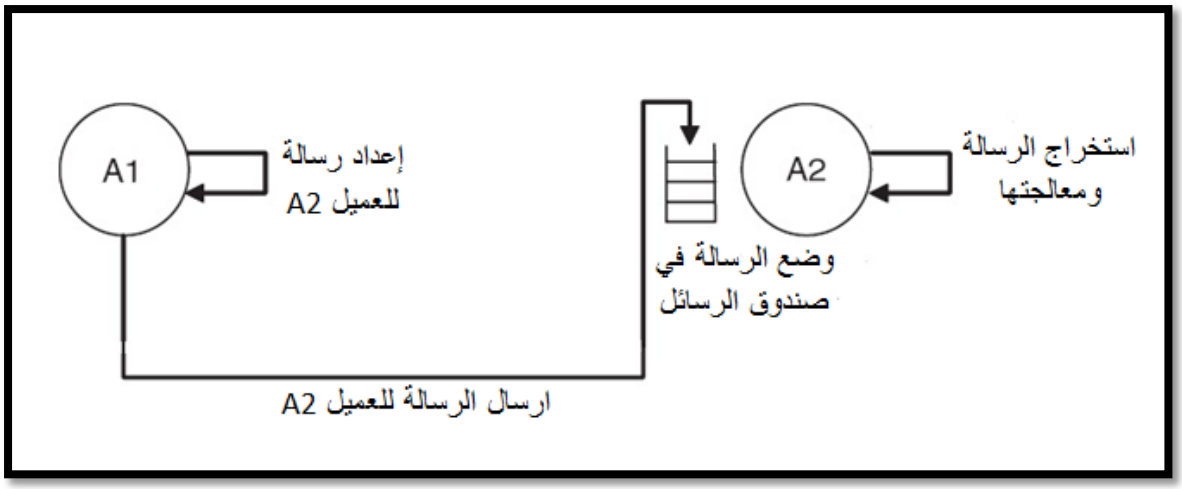
- Name: هو اسم الحالة.
- State: هو المهمة(Behaviour) التي تمثل الحالة
- s1: الحالة الاصلية(الحالة التي سينتقل منها العميل).

- s2:الحالة الهدف(الحالة التي سينتقل اليها العميل).
- event: القيمة التي في حال الحصول عليها يطبق هذا الإنتقال.
- **toBeReset**: يضم المهام التي يجب اعادة تهيئتها قبل اعادة تنفيذها.

```
protected void setup() {
	FSMBehaviour sampleFSM = new FSMBehaviour(this);
	sampleFSM.registerFirstState(new OneShotBehaviour(this){
		public void action() {
			// perform operation X
		}
		public int onEnd() {
			return (operation X successful ? 1 : 0);
		}
	}, "X");
	sampleFSM.registerState (newOneShotBehaviour(this){
		public void action() {
			// Perform operation Y
		}
	}, "Y");
	sampleFSM.registerLastState (new OneShotBehaviour(this){
		public void action() {
			// Perform operation Z
		}
	}, "Z");
	sampleFSM.registerTransition("X", "Y", 1);
	sampleFSM.registerTransition("X", "Z", 0);
	sampleFSM.registerDefaultTransition("Z", "X",
			new String[]{"X","Z"});
}
```

7. التواصل بين العملاء:

الفئة ACLMessage التي تتضمنها المنصة Jade تمثل لغة التواصل بين العملاء (Agent Communication Language -ACL) والتي تسمح بإمكانية ارسال الرسائل بين العملاء، حيث يملك كل عميل صندوق رسائل يعمل عمل البريد الالكتروني، فهو يخزن الرسائل التي تم استقبالها لغرض معالجتها من قبل العميل كما يبين المخطط التالي:



تستعمل المنصة Jade رسائل متوافقة مع شروط FIPA ككائنات مستنسخة من الفئة jade.lang.acl.ACLMessage وتتضمن الحقول التالية:

- المرسل: هذا الحقل يملأ تلقائيا عند الارسال.
- مجموعة مستقبلي الرسالة: الرسالة يمكن ان ترسل لعدة عملاء في نفس الوقت.
- هدف الاتصال: يتواصل العملاء للعديد من الاهداف كإعلام العميل بأمر ما او الرد على استعلام أو طلب خدمة ....الخ.
- محتوى الرسالة.
- عدة حقول اختيارية مثل اللغة، عنوان الرد، اخر اجل للرد....الخ.

أ- ارسال الرسائل:

على العميل الذي هو بصدد ارسال رسالة أن ينشئ كائن جديد من الفئة ACLMessage ثم يملأ عناصرها بالبيانات اللازمة ثم أخيرا يستدعي الدالة Agent.send(). في المثال التالي يقوم عميل بإعلام عميل اخر عن حالة الطقس عن طريق رسالة اخبارية(INFORM).

```
public class MyAgent extends Agent {
	protected void setup() {
		ACLMessage message= new ACLMessage(ACLMessage.INFORM);
		message.addReceiver(new AID("Agent2", AID.ISLOCALNAME));
		message.setContent("Today it’s raining");
		send(msg);
	}
}
```

ب- إنتظار وصول رسالة:

يمكن ان نجعل العميل يقوم بالانتظار النشط بهذا الشكل:

```
public class Agent2 extends Agent{
	protected void setup() {
		addBehaviour(new OneShotBehaviour(this) {
			public void action() {
				ACLMessage message =null ;
				while (message == null){
					message = receive() ;
				}
				//traitement
			}
		});
	}
}
```

لكن هذه الطريقة غير فعالة وتستهلك موارد الجهاز لذى الأفضل ان نقوم بتجميد المهمة الى غاية ورود رسالة وذلك عن طريق الدالةblocking()

```
public class Agent2 extends Agent{
	protected void setup() {
		addBehaviour(new OneShotBehaviour(this) {
			public void action() {
				ACLMessage message  = receive();
				if (message  == null) block();{
					// Process the message
				}
			}
		});
	}
}
```

أو عن طريق الدالةblockingReceive() كما يلي:

```
public class Agent2 extends Agent{
	protected void setup() {
		addBehaviour(new OneShotBehaviour(this) {
			public void action() {
				ACLMessage message  = receive();
				if (message  == null) blockingReceive ();{
					// Process the message
				}
			}
		});
	}
}
```

ت- اختيار رسالة من صندوق الرسائل:

في الغالب من الاحيان يتواصل العميل مع اكثر من عميل في نفس الوقت، وبالتالي يستقبل اكثر من رسالة من اكثر من عميل في آن واحد، لذى كان على كل مهمة اختيار الرسالة التي تخصها فقط، ويتم ذلك بإنشاء كائن من الفئة jade.lang.acl.MessageTemplate وهو عبارة عن نموذج رسالة(Template Message) محاكي للرسالة الهدف، ومن ثم المقارنة مع الرسائل الواردة بهدف الحصول على رسالة تتوافق مع هذا النموذج. وهذا مثال على ذلك:

```
MessageTemplate Template=
            MessageTemplate.MatchPerformative(ACLMessage.INFORM);
ACLMessage Message= myAgent.receive(Template);
```

8. هجرة العملاء:

العميل المهاجر او المتنقل بالإنجليزية agent mobile هو عميل يستطيع التنقل عبر الشبكة من منصة الى اخرى، حيث انه يوقف تنفيذ مهماته عند بدء انتقاله(محتفظا بجميع خصائصه وقيمه) ويستأنف التنفيذ بعد انتقاله من نقطة توقفه الأولى.

ويتعلق الأمر بثلاث دوال توفرها الفئة Agent وهي doMove() وهي التي تضمن انتقال العميل وتأخذ كمُدخلات كائن destination ويمثل وجهة العميل و يوجد على المنصة ضمن الحزمة jade.core فئتان توفران نوعان من هذا الكائن، الأولى هي ContainerID وتستعمل مع العميل الذي ينتقل من container الى اخرى على نفس المنصة التي يعمل عليها، أما الثانية فهي PlatformID وتستعمل عند انتقال العميل الى container على منصة غير المنصة التي يعمل عليها العميل.

أما الدالتين الباقيتين فهما beforeMove() وafterMove()، كما هو واضح من اسم كل دالة فإن واحدة منهما تحدد المهام التي يقوم بها العميل قبل انتقاله والأخرى تحدد المهام التي يقوم بها بعد الإنتقال.

وكمثال عن الانتقال على نفس المنصة نأخذ المثال التالي، وهو عبارة عن عميل ينتقل الى container فرعية، وبما أننا على نفس المنصة اذا سنستخدم ContainerID.

```
protected void setup(){
	//name of the container
	String containerName = "Container-1";
	//create object with ContainerID type
	ContainerID destination = new ContainerID();
	// Initialize the destination object
	destination.setName(containerName);
	// start the migration
	doMove(destination);
}
//@Override
protected void beforeMove() {
	System.out.println("try to move...");
}
// @Override
protected void afterMove(){
	System.out.println("move done");
}
```

والآن لنأخذ مثال عن الانتقال الى منصة اخرى غير التي يعمل عليها العميل، أي أننا سنستخدم PlatformID.

```
protected void setup() {
	//Build the AID of the corresponding remote platform's AMS
	AID remoteAMS = new AID("ams@remotePlatform:1099/JADE", AID.ISGUID);
	//Specify the MTP by setting the transport address of the remote AMS
	remoteAMS.addAddresses("http:// remotePlatformaddr:7778/acc");
	// create and initialize the destination object
	 PlatformID destination = new PlatformID(remoteAMS);
	// start the migration
	 doMove(destination);
}
//@Override
protected void beforeMove() {
	System.out.println("try to move...");
 }
// @Override
protected void afterMove(){
	System.out.println("move done");
}
```

انتهى...

في الختام

أرجو من الله أن أكون قد وفقت في تقديم هذا العمل خالصا لوجهه الكريم وعلى الوجه الذي يحب ويرضى.