Apprendre Java avec BlueJ, une autre approche (1ère partie)

Reference : *Objects First with Java* – A Practical Introduction Using BlueJ
David Barnes & Michael Kölling

Auteur : Boichat Jean-Bernard Email : jean-bernard@boichat.ch Version de cet article : 1.1.1

Publication récente : *Apprendre Java et C++ en parallèle – 4ième édition*
http://www.eyrolles.com/Informatique/Livre/apprendre-java-et-c-en-parallele-9782212124033
http://www.boichat.ch/javacpp/

Introduction à BlueJ

Même avec plusieurs années d'expérience et de vagabondage avec Java (et C++), j'ai été surpris par ce livre et l'approche BlueJ. Ce fut un peu comme un coup de foudre, à retardement, car j'aurais dû déjà le rencontrer bien avant, quand j'étais plus jeune !

Ayant juste terminé la 4ième édition de mon livre et ayant commencé un nouveau travail avec un collègue devant acquérir des connaissances de base en Java, j'ai vite compris que BlueJ pourrait nous aider tous les deux (oui, moi aussi : il n'est jamais interdit de se remettre en question).

Commencer par une classe en Java dans le premier chapitre (au lieu d'un classique « Hello World ») est à la fois surprenant et réconfortant (oui, c'est possible). Bien sûr que l'approche UML vevrait être la meilleure, mais mon dieu, que c'est lourd pour un débutant.

Au contraire de Bruce Eckel et de son livre *Thinking in Java*[1], plus direct avec des exemples, David Barnes & Michael Kölling choisissent une approche plus « intellectuelle » et une présentation magnifique (le format et le look de l'ouvrage est une merveille qui nous aidera, un peu, à justifier son prix excessif).

Oui, si on est un étudiant fauché, comme souvent, nous pourrons trouver BlueJ sur Internet[2] et avec une version encore plus récente que celle de la dernière édition du livre, la version 2.5.0 à ce jour. Il existe aussi un tutorial en français[3], mais comme il fait référence au JDK 1.3, il n'est sans doute à consulter qu'en dernier recours.

Comme dans beaucoup de livres ou de sites Internet, et c'est le cas ici, les auteurs n'ont pas cherché à donner des détails d'installation. Il faut savoir se débrouiller. Ce n'est pas trop mon approche, car j'ai souvent donné moi-même plus d'importance à l'installation, soit pour les débutants ou encore à cause des nombreuses configuration de XP, Vista, Linux et des machines plus ou moins performantes. L'avantage ici, au contraire d'un livre, est que nous peut mettre à jour rapidement un article sans devoir attendre la prochaine édition. J'essaierai donc de donner au mieux possible les instructions d'installation, d'y ajouter des commentaires bienvenus de lecteurs ou encore des extensions.

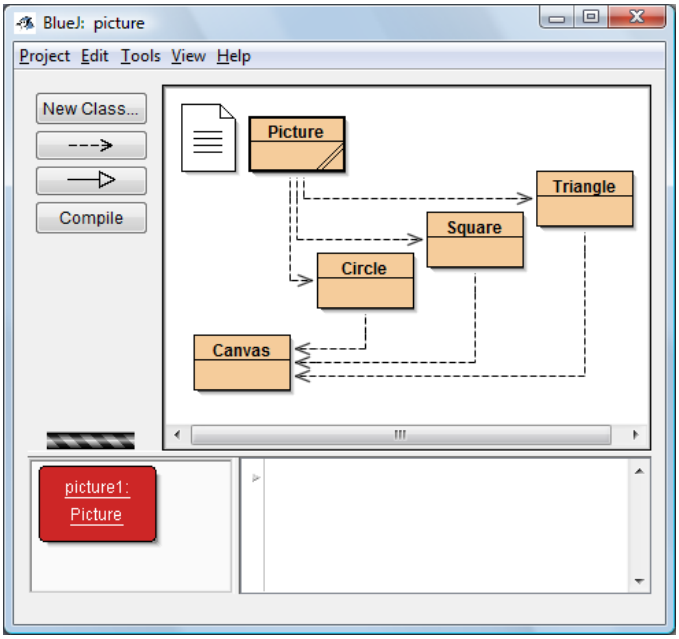
[1] http://www.mindview.net/Books/TIJ/ et http://mindview.net/Books/TIJ4
[2] http://www.bluej.org/
[3] http://www.bluej.org/tutorial/tutorial_french.pdf

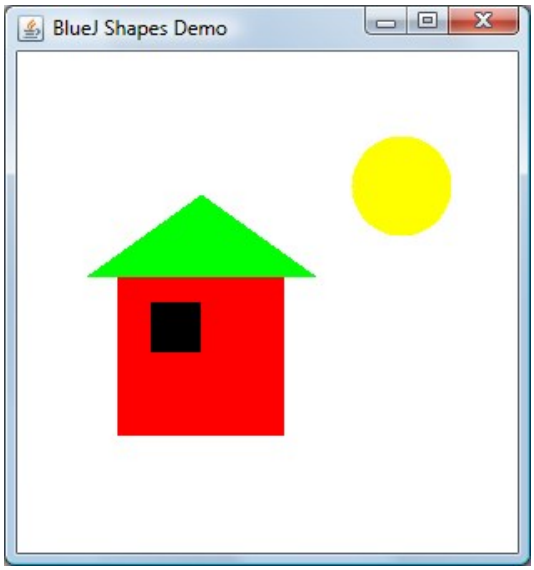
BlueJ et l'orientation objet

BlueJ est un environnement de développement (IDE) pour la programmation Java et avec une conception orientée objet. BlueJ est lui-même écrit en Java.

Comme pour d'autres IDE (Netbeans[4] et Eclipse[5], par exemple), il faut d'abord créer un projet et ensuite des classes, des programmes de tests et finalement des applications.

Ici tout ce fait de manière graphique et visuelle :





Au contraire d'un IDE classique, où l'on commencera par entrer ou copier/coller du code Java dans un éditeur, nous aurons un ensemble d'outils pour préparer notre travail avant édition ou exécution.

Une grande partie des fonctions se fait avec la souris. De plus, au contraire d'un IDE Java traditionnel, nous pourrons instancier des objets de classes, concept clé de la programmation objet.

Cette approche essentielle est non seulement intéressante pour des débutants, mais surtout pour acquérir correctement les concepts de base de la programmation objet.

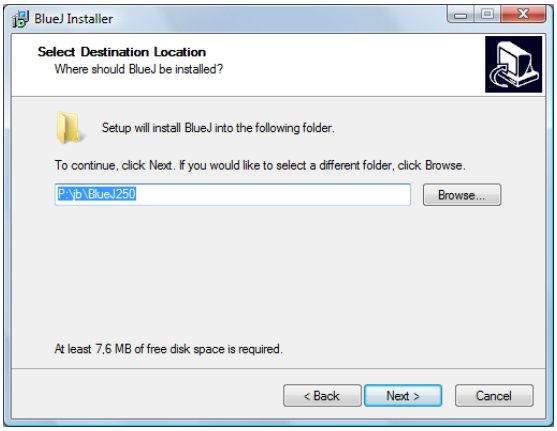
[4] http://www.netbeans.org/
[5] http://www.eclipse.org/

Installation de BlueJ (Windows ou Vista)

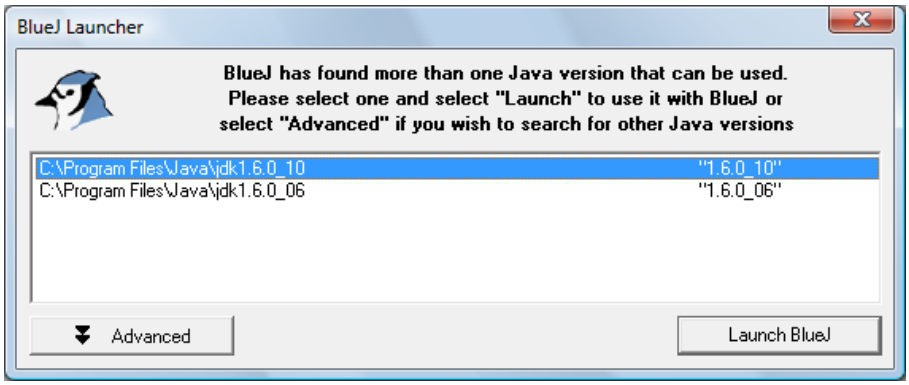
Nous avons choisi la version la plus récente, la version 2.5.0 citée ci-dessus et téléchargée depuis http://www.bluej.org/download/download.html

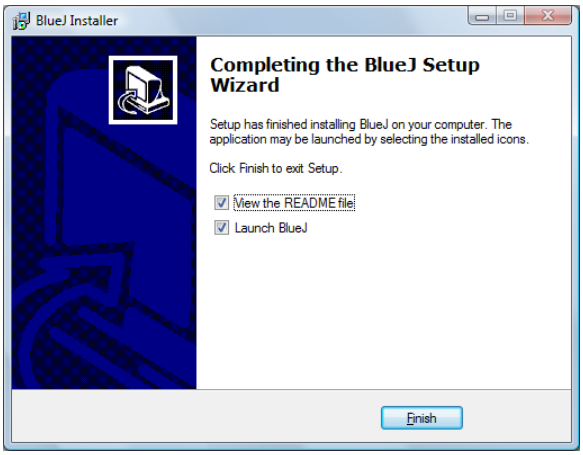
Des versions pour MacOS X et autres systèmes comme Linux (avec un fichier `.jar` exécutable) existent aussi.

Il faudra donc exécuter le fichier d'installation **`bluejsetup-250.exe`** et indiquer un endroit où l'installer, comme par exemple ici `P:\jb\BlueJ250` :



Au démarrage nous pourrions recevoir cette information, qui nous demande quelle version de Java doit être utilisée :





Même si nous n'avons pas les dernières versions (JDK 6 conseillé), une version 1.4 irait aussi.

Pour terminer, l'indication de l'installation terminée avec succès sera présentée :

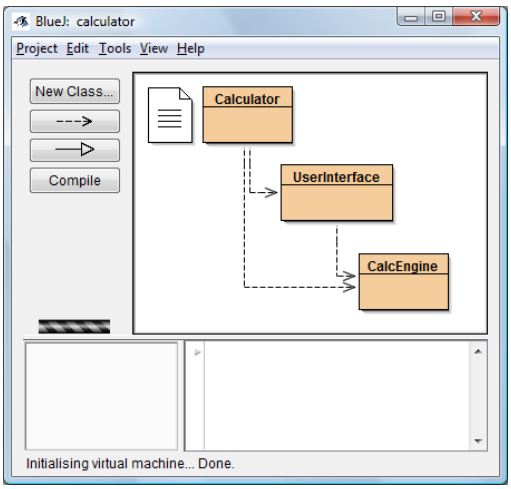
BlueJ – La simplicité

Bien des références et des articles existent sur la toile comme par exemple :

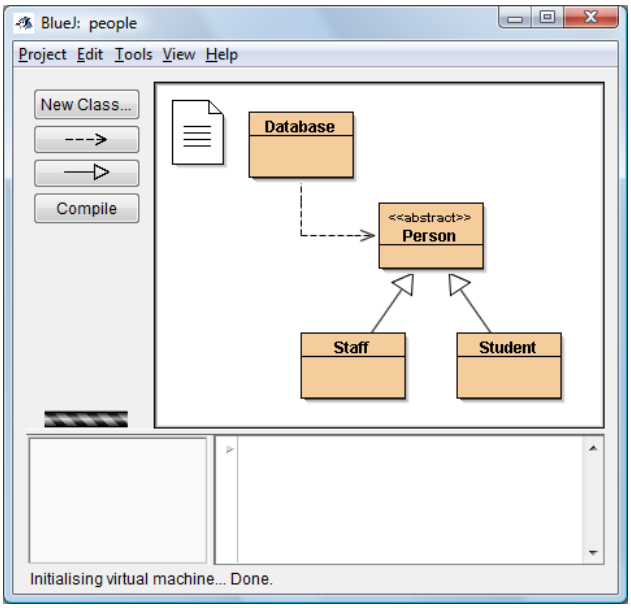
- BlueJ sur Wikipedia[6]
- BlueJ sur Sun Developer Network[7]

On nous parle de simplicité d'interface, de facilité d'installation et d'utilisation, mais c'est sans doute son aspect visuel et son orientation objet que nous retiendrons d'abord. Un autre aspect sans doute : la rapidité sur de petites machines avec d'anciens CPUs ou peu de RAM.

Si nous démarrons BlueJ avec le menu traditionnel, nous obtiendrons le projet utilisé en dernier, comme par exemple :



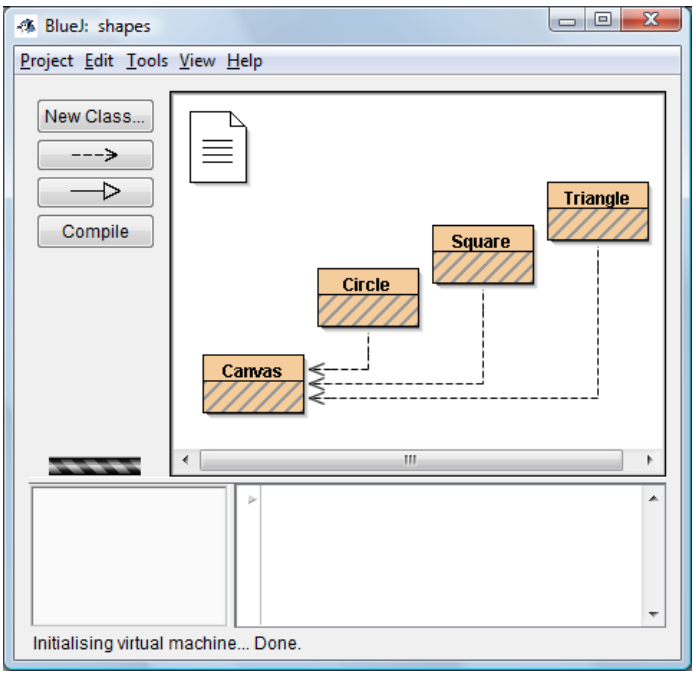
Nous pouvons aussi démarrer un projet depuis l'explorateur, par exemple depuis **`..\BlueJ250\examples\people`**, en double-cliquant sur un projet avec l'extension **`.pkg`**, ici **`people.pkg`** :



Ce dernier est l'exemple que nous trouvons au début du tutorial de Michael Kölling[8]

[6] http://en.wikipedia.org/wiki/BlueJ
[7] http://java.sun.com/features/2002/07/bluej.html
[8] http://www.bluej.org/doc/documentation.html

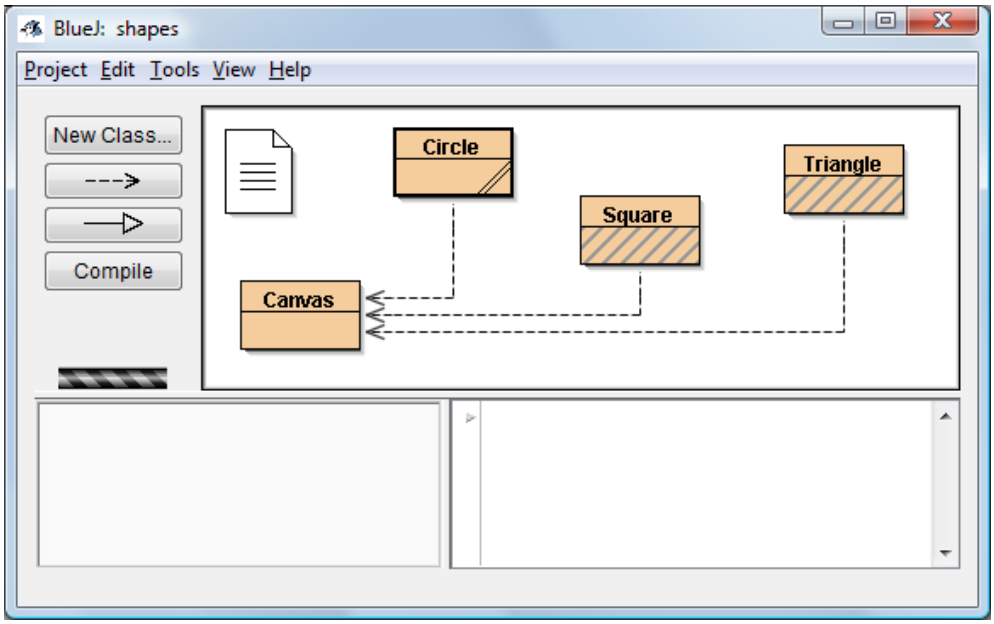
Ce n'est sans doute pas le meilleur exemple à choisir pour débuter. Nous pensons que le projet **shapes** (formes) est plus approprié pour une première mise en main.



Nous commencerons donc par le projet **`bluej.pkg`** dans le répertoire **`..\BlueJ250\examples\shapes`**.

Ce projet se trouve, avec raison d'ailleurs, dans le premier chapitre du livre *Objects First with Java* cité ci-dessus (voir aussi le README.TXT en anglais dans ce même répertoire) :

Une des premières choses que nous constaterons sans doute est qu'il est possible de déplacer nos classes et de les réorganiser dans notre plan de travail :

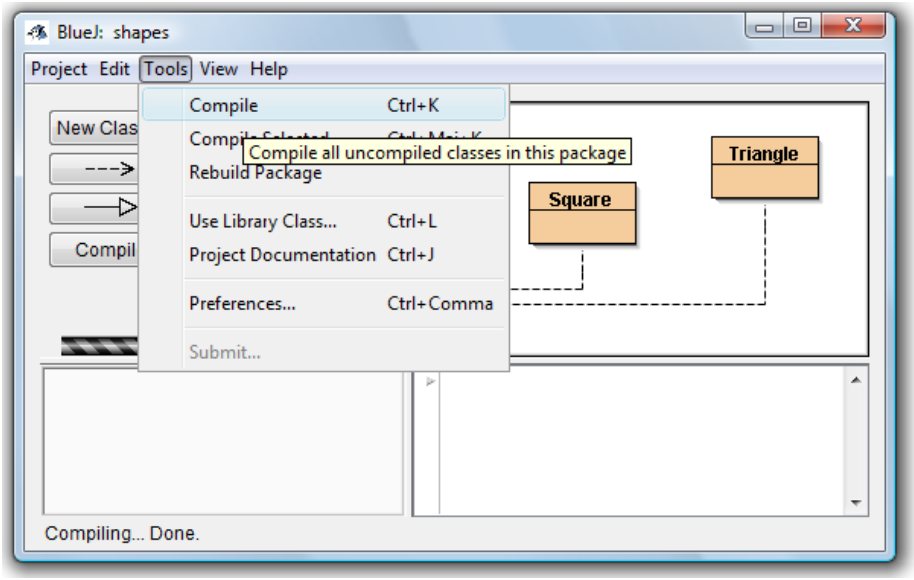


Nous remarquerons rapidement les différentes présentations des classes : ici les classes **Canvas** et **Circle** sont compilées et pas les classes **Square** et **Triangle** (striées en bleu). La classe **Circle** (deux barres) est la classe sélectionnée avec un clic de souris pour d'autres opérations.

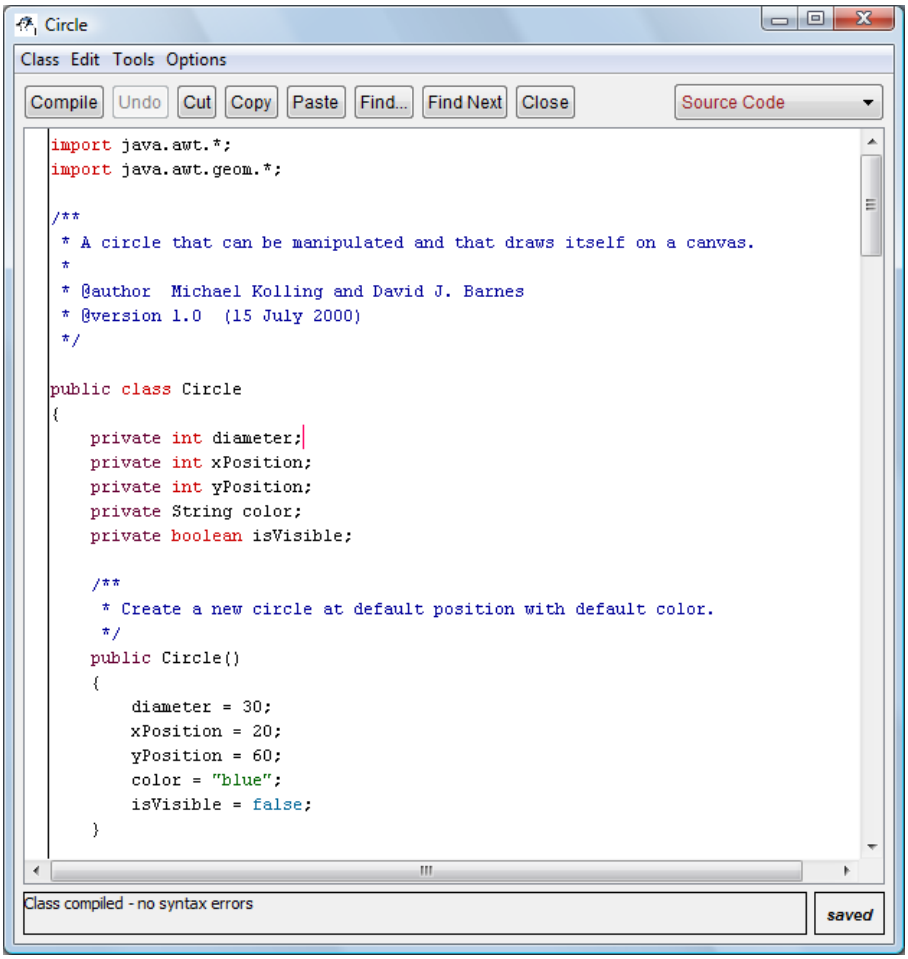
La classe **Canvas** est un peu plus compliquée, avec des classes internes et un Singleton. Ces concepts devraient être expliqués dans la plupart des ouvrages consacrés à Java. Cette classe nous permettra d'afficher nos objets et nous y reviendrons.

Les classes Java compilées ont l'extension **.class** et en effaçant **Square.class** et **Triangle.class**, directement dans le répertoire, nous obtiendrons cet effet. Nous verrons aussi que, dès que l'éditeur est ouvert sur une classe et dès qu'une modification est apportée dans le code source, la classe deviendra striée (donc à recompiler).

Nous indiquerons déjà, qu'avec le menu Tools (outils), nous pourrions aussi compiler tout le projet ou une classe individuelle :



Comme dans le chapitre 1 d'« *Objects First with Java* », nous commencerons par la classe **`Circle`** (cercle) et pourquoi pas double-cliquer sur son diagramme de classe :




```
import java.awt.*;
import java.awt.geom.*;

/**
 * A circle that can be manipulated and that draws itself on a canvas.
 *
 * @author  Michael Kolling and David J. Barnes
 * @version 1.0  (15 July 2000)
 */

public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new circle at default position with default color.
     */
    public Circle()
    {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }
```

L'éditeur de BlueJ est à la hauteur de sa simplicité. Les boutons sont faciles à comprendre. Après édition, nous pourrons compiler le code (« Compile »). Le bouton « Close » fermera la fenêtre en sauvant le code : il n'y a pas de message d'avertissement.

Au premier coup d'oeil, pour la classe **Circle**, c'est assez clair, quoique pas trop générique : ce cercle possède une position, un diamètre, une couleur et une indication s'il est visible.

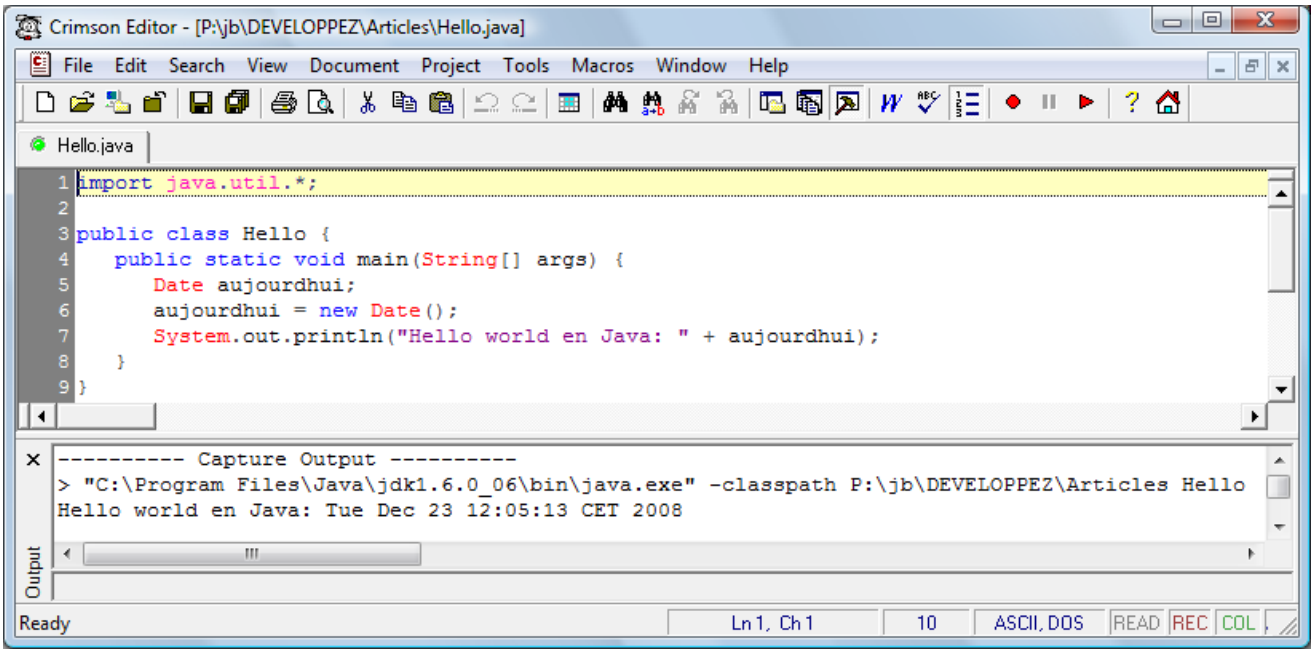
Mais l'aspect le plus déroutant pour les habitués de Java:

- Il n'y a pas de `main()` !
- Comment pouvons nous exécuter « quelque chose » ?

Exécution d'un programme

Nous sommes vraiment dans le monde orienté objet. Cela me rappelle un de mes premiers cours en C++ ou nous avions commencé par une introduction « forcée » à Smalltalk[9]. Comme avec BlueJ, la première action concrète avait été d'instancier un objet.

Par contre, dans le premier chapitre de mon livre, le premier exemple en Java est écrit ainsi (éditeur *Crimson* qui intègre tous les outils de développement Java, C++, C#, et make) :



Nous dirons que le code aurait pu être « pire » :

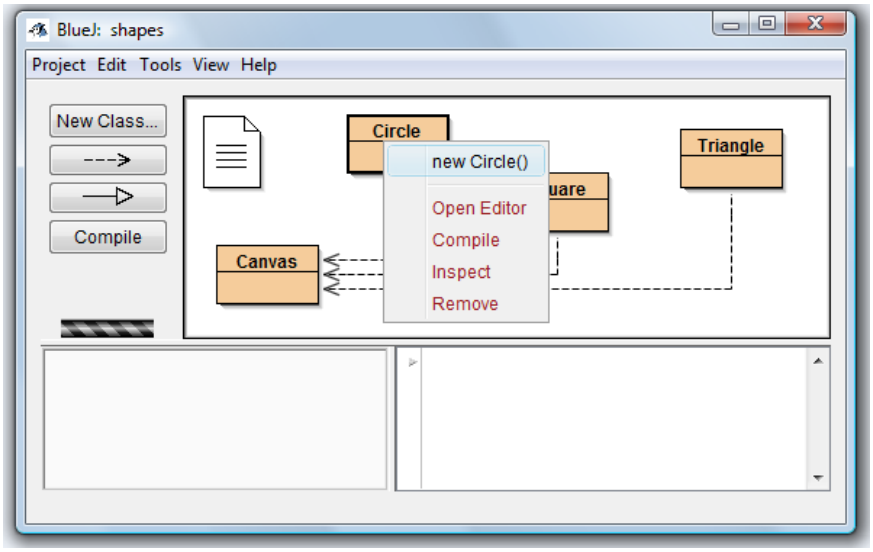
```
public class Hello2 {
   public static void main(String[] args) {
      System.out.println("Hello world en Java: " + new java.util.Date());
   }
}
```

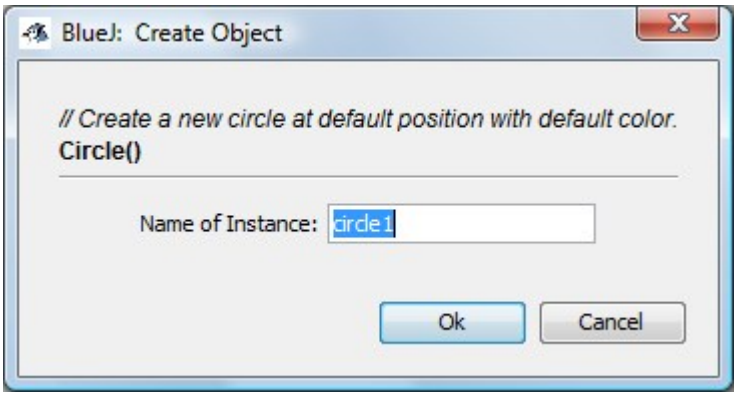
Nous n'avons jamais instancié de classe `Hello` (ou `Hello2`) et l'instanciation de la classe `Date` ne nous aide qu'à imprimer la date d'aujourd'hui.

[9] http://fr.wikipedia.org/wiki/Smalltalk

Pour notre cercle, c'est évident : nous aimerions le voir, le dessiner et jouer avec. Il nous faut donc instancier la classe `Circle` pour en faire un objet utilisable et visible.

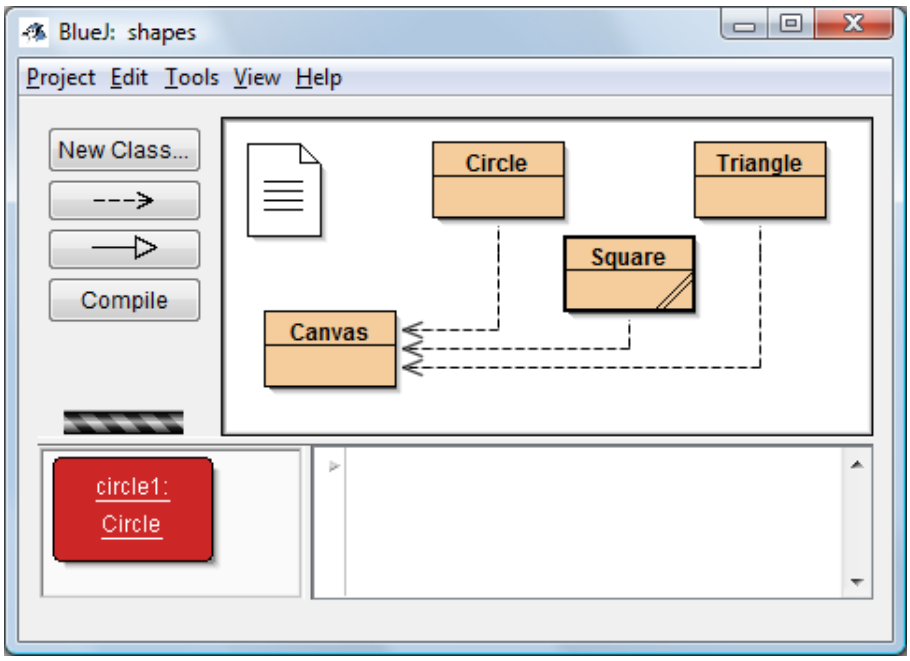
Dès que la classe `Circle` est compilée, et si nous cliquons sur cette classe avec le bouton de droite de la souris, nous obtiendrons ceci :



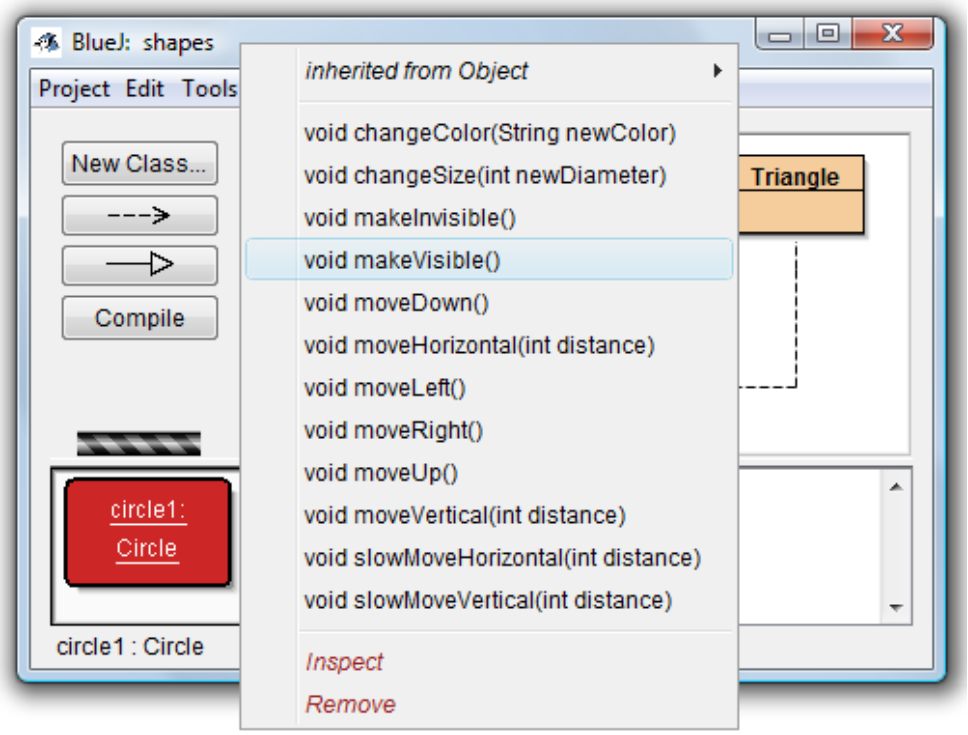


En créant une nouvelle instance (objet) de la classe `Circle`, il nous faut lui donner un nom pour l'identifier et cliquer sur Ok.

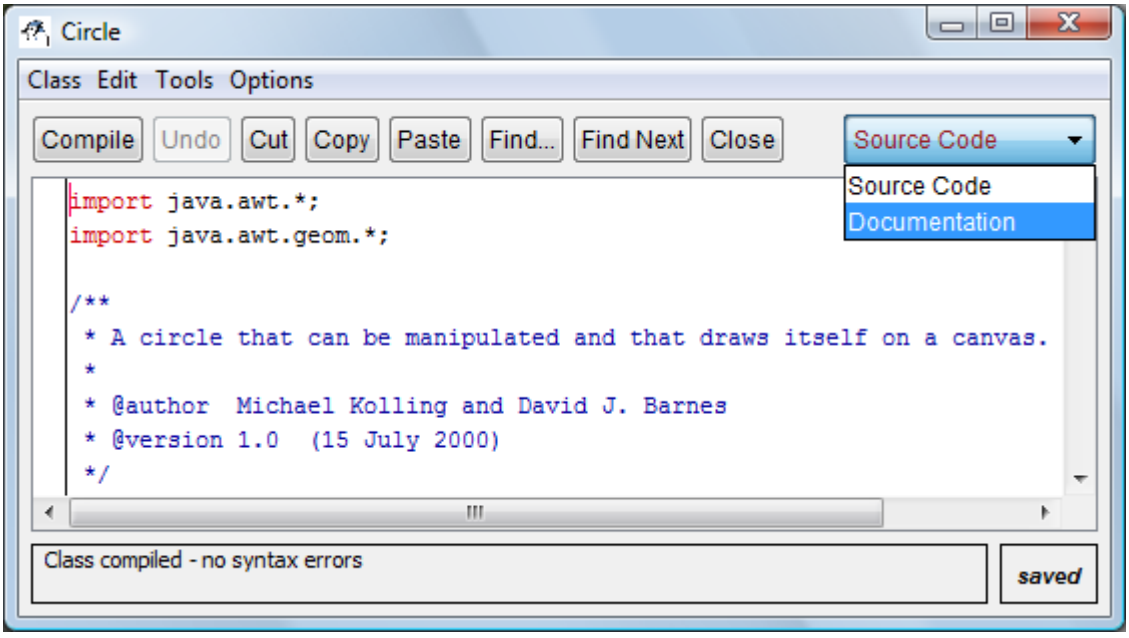
L'instance de la classe `Circle`, c'est à dire l'objet `circle1` va apparaître à présent dans la fenêtre inférieure de gauche.



A nouveau en utilisant le bouton de droite de la souris, et sur l'objet `circle1` de la classe `Circle` (en rouge), nous découvrirons toutes les méthodes publiques disponibles pour cette classe :



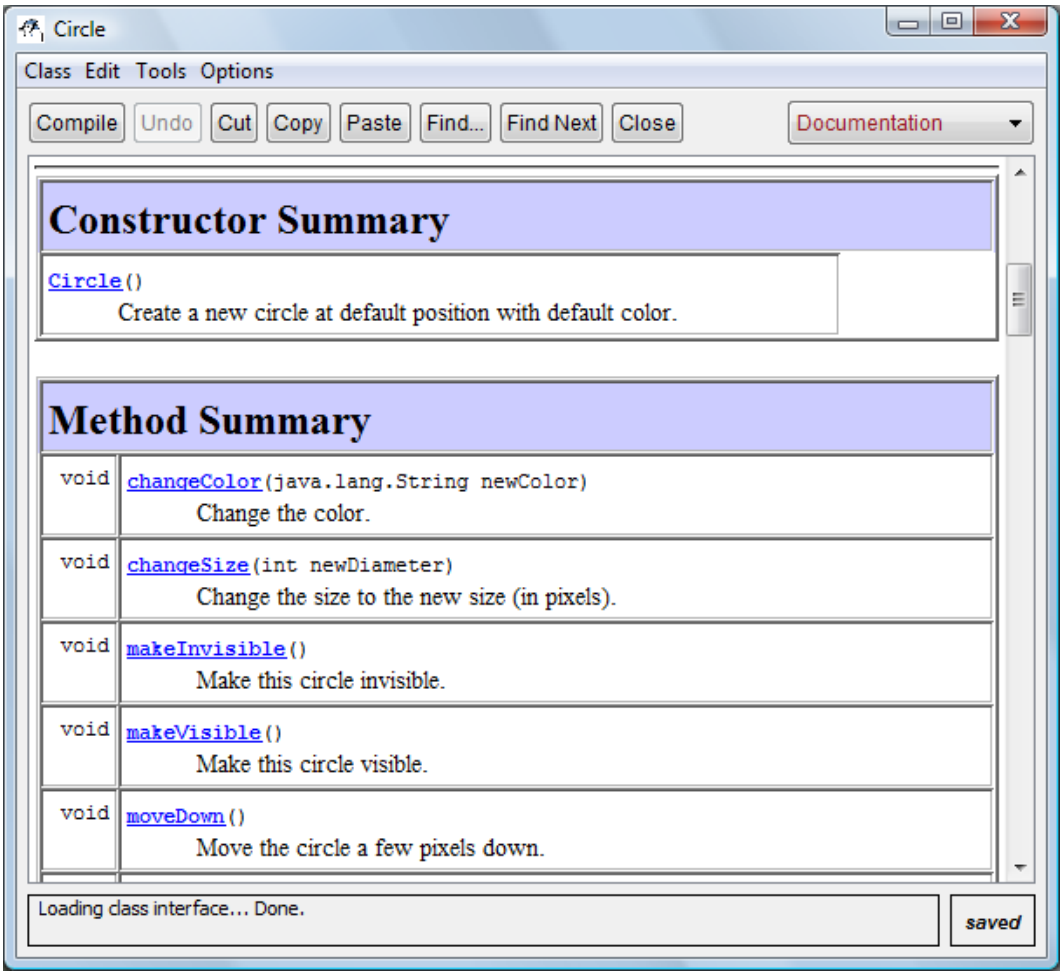
C'est sans doute le bon moment de revenir à l'éditeur et sa génération de la documentation :



La documentation de la classe nous sera présentée. Elle est générée par l'outil `javadoc`[10] et dépendra de l'information entrée par le programmeur. Des outils comme Netbeans permettent une génération semi-automatique de texte comme décrit dans l'annexe E de mon livre.

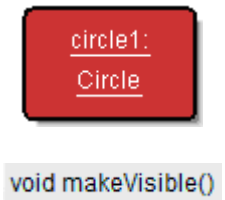
[10] http://fr.wikipedia.org/wiki/Javadoc

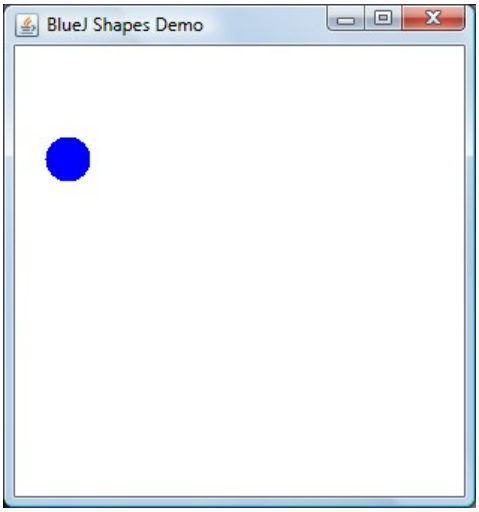
En consultant la documentation :



Nous allons rapidement constater que la méthode `makeVisible()` devra être utiliser pour obtenir un premier résultat :

En reprenant le bouton de droite de la souris, et sur l'objet `circle1` dessiné en rouge de la classe `Circle`, et en sélectionnant la méthode `makeVisible` (rendre visible) une nouvelle fenêtre va apparaître avec notre premier cercle bleu dessiné dans une fenêtre Widows.





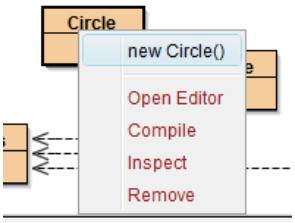
C'est effectivement une toute autre approche que si nous avions développé nous-même cette application avec un outil tel que Netbeans et créé tout l'environnement depuis un point d'entrée `main()` en passer par une fenêtre `Window` et un éventuel `Canvas` ou encore un `JPanel`.

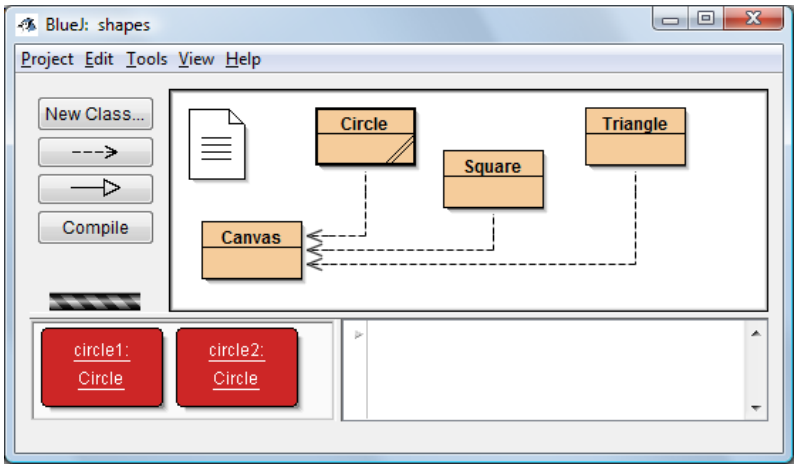
Quelles sont les questions que nous pourrions nous poser ? En voici une petite liste parmi d'autres :

- Où se trouve le point d'entrée `main()` traditionnel ?
- Est-ce que notre cercle bleu est de la bonne grandeur et comment le repositionner ?
- Peut-on ajouter un second cercle ?
- Comment modifier le code ?

Dans le répertoire `..\BlueJ250\examples\shapes`, il y a bien 4 fichiers `.java` correspondants à nos 4 classes `Circle`, `Square`, `Triangle` et `Canvas`, mais aucune d'elle ne contient une entrée `main()`. Il y donc un mécanisme dans BlueJ qui va instancier quelque part une classe `Window`, contenant un `Canvas` et un ou plusieurs formes instanciées.

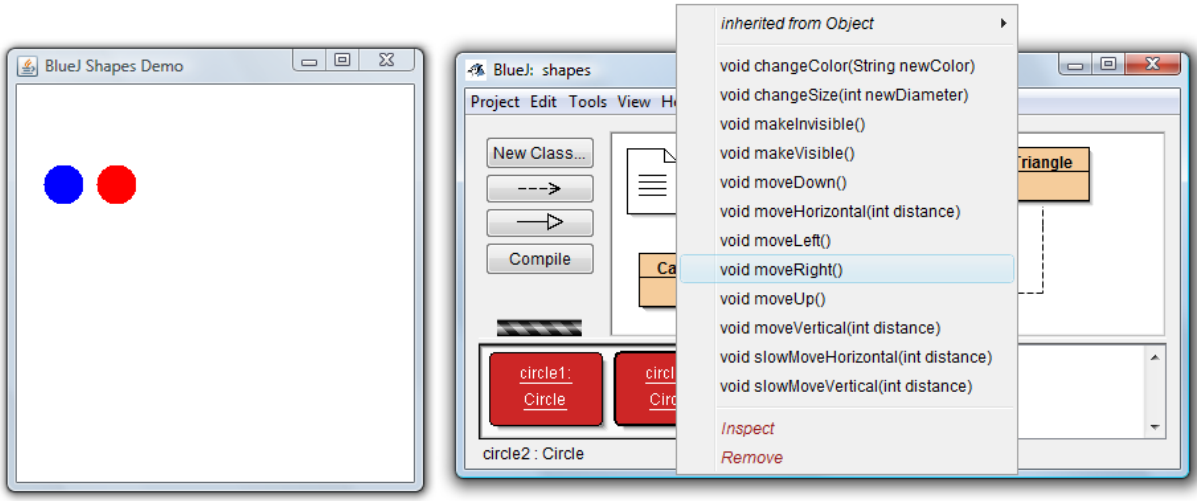
Pour déchiffrer la bête, nous allons instancier deux objets de la classe `Circle` : `circle1` et `circle2` et toujours de la même manière, avec le bouton de droite de la souris :





Comme le `new Circle()` pour le `circle2` appelle le même constructeur, nous aurons les deux cercles de même couleurs, de même taille et au même endroit : question visibilité, ce n'est pas l'idéal.

Pour obtenir l'effet ci-dessous, il faudra appliquer un certain nombre de méthodes sur les objets `circle1` et `circle2` : bouton de droite de la souris sur les instances marqué en rouge.



Les opérations auront été ici :

- `moveRight()` (déplacement à droite) de l'objet `circle1`
- `moveRight()` (déplacement à droite) de l'objet `circle1`, une seconde fois
- `changeColor()` avec "`red`" (rouge) pour l'objet `circle1` (c'est la seule méthode qui demande un paramètre, et, pour une chaîne de caractères, il faut impérativement mettre des guillemets)
- et finalement un `makevisible()` pour chacun des deux objets pour voir le résultat

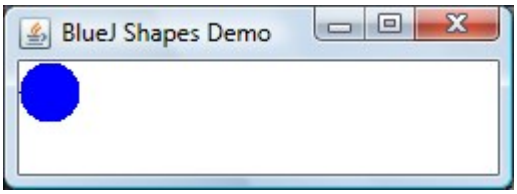
Pour le positionnement des objets, il y a plusieurs méthodes, comme par exemple :

- Nous regardons le code et l'API du JDK
- Nous essayons de modifier le code dans `Circle.java` comme ceci :

```
public Circle()    {
    diameter  = 30;
    xPosition = 0; //20;
    yPosition = 0; //60;
```

En recompilant le code (bouton «Compile ») dans l'éditeur et en créant à nouveau objet, nous aurons alors le résultat présenté ainsi :

En recompilant le code (bouton «`Compile` ») dans l'éditeur et en créant à nouveau objet, nous aurons alors le résultat présenté ainsi :



Les formes sont donc positionnées en haut à gauche et non pas par rapport au centre du cercle. Il nous reste à présent à élucider l'entrée `main()` dont voici quelques détails et en reprenant notre exemple avec nos deux cercles :

- **Avant** un `makevisible()` rien ne se passera, mais des corrections et des adaptations sur les attributs de chaque objet se produiront suivant les méthodes appliquées.

- Un `moveRight()`, par exemple, va déplacer la forme à droite de 20 pixels en appelant la méthode `moveHorizontal(20)`. Le code de cette méthode :

  ```
  erase();
  xPosition += distance;
  draw();
  ```

  nous montre que l'objet sera effacé (`erase()`) et redessiné (`draw()`). Mais rien ne se passera si l'objet n'est pas visible (`visible` égal à `true`).

Nous terminerons par le plus intéressant, sans aucun doute, le `makevisible()`. Que se passe-t-il exactement ?

```
public void makeVisible() {
    isVisible = true;
    draw();
}
```

et ensuite :

```
private void draw() {
   if (isVisible) {
      Canvas canvas = Canvas.getCanvas();
      canvas.draw(this, color,
                  new Ellipse2D.Double(xPosition, yPosition, diameter, diameter));
      canvas.wait(10);
   }
}
```

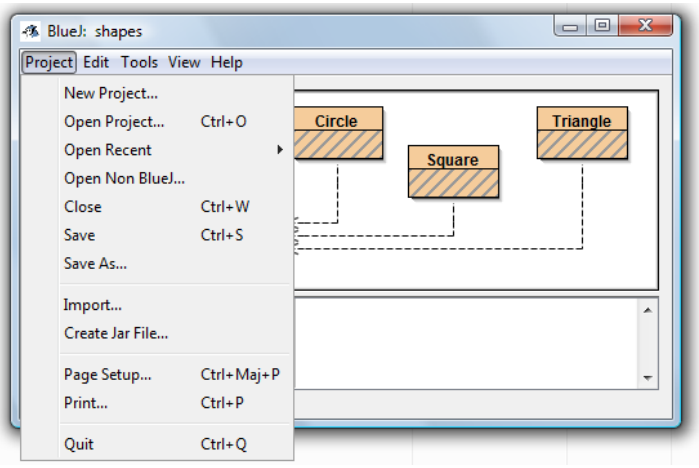
Le `getCanvas()` retourne une instance d'un `Canvas` au moyen d'un Singleton (voir mon ouvrage *Apprendre Java et C++ en parallèle* ou encore http://fr.wikipedia.org/wiki/Singleton_(patron de_conception) par exemple). Le `Canvas` existe peut-être déjà, sinon il est créé. Il n'y a qu'une seule instance possible et c'est pour cette raison que deux ou plusieurs objets peuvent être déposés dans un seul et unique `Canvas`. Dans le constructeur de ce dernier une `JFrame` est aussi créée : elle correspond à la fenêtre Window que nous nous voyons effectivement avec nos dessins instanciés et visibles.

Et il n'y a pas d'entrée `main()` du tout : l'application … c'est BlueJ ! C'est elle qui appelle la méthode `makevisible()` !

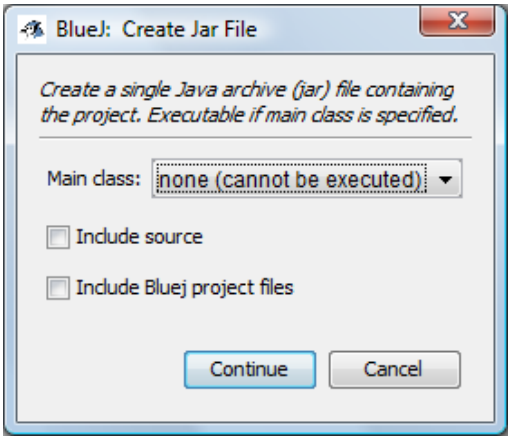
`main()`, `.jar`

Nous terminerons encore une fois avec le `main()`, les fichiers `.jar` et le débogueur intégré de BlueJ. Nous reviendrons sur ces trois sujets dans le second article.

Avec le menu
`Project / Create Jar File…` :



Il est possible de créer un fichier `.jar` (une archive) contenant toutes nos classes de notre projet :



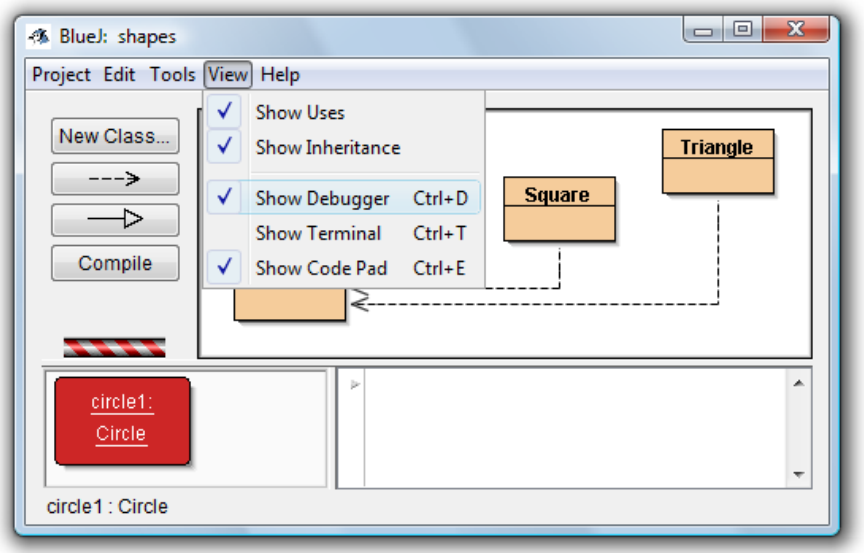
Nous remarquons qu'il n'y a pas de `main()` (`none`) et que ce fichier `.jar` ne nous aidera pas à exécuter une application Java. Nous pourrons aussi intégrer notre code source et les fichiers projet dans cet archive, c'est pratique.

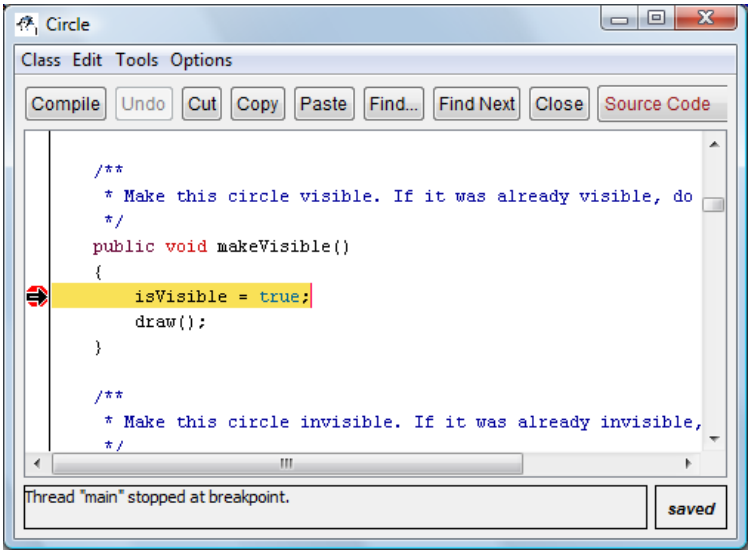
Dans le second article consacré à Netbeans, cous créerons une nouvelle classe Java contenant une entrée `main()`. Nous pourrions d'ailleurs très bien faire ce travail ici, sans Netbeans.

A titre d'information, nous indiquerons qu'un fichier `.jar` peut être affiché en utilisant un outil comme Winzip ou 7-Zip (utilisé depuis la 4-ième édition d'*Apprendre Java et C++ en parallèle* : http://www.7-zip.org/ et que je conseille absolument sur un PC Windows ou Vista). Dans le répertoire `META-INF`, il y a un fichier `MANIFEST.MF` : c'est lui qui définit l'entrée principale `main()` du fichier `.jar` (voir par exemple : http://fr.wikipedia.org/wiki/Jar).

Le débogueur de BlueJ

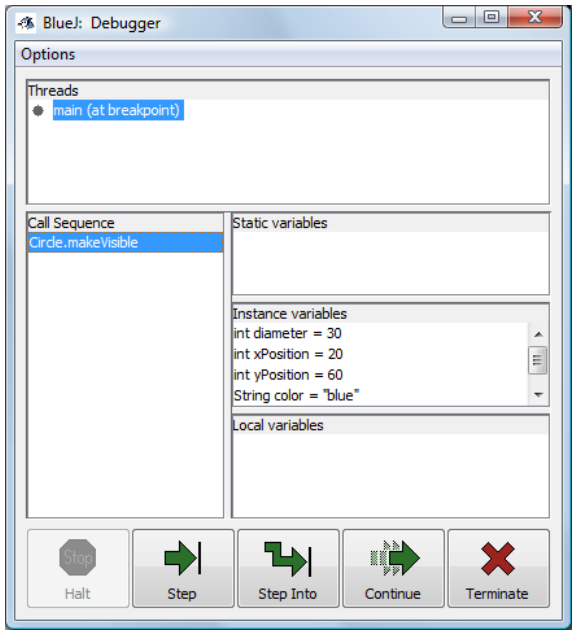
Pour rendre le débogueur active il faut l'activer avec le menu `View / Show Debugger` :





Pour définir un point d'arrêt, il faut cliquer dans la colonne de droite (point rouge).

En activant la méthode `makevisible()` comme décrit ci-dessus, le programme s'arrêtera au point d'arrêt défini et nous donnera les valeurs des attributs.



Suggestions pour des exercices

- Le débutant pourra jouer à souhait avec ce code en ajoutant, par exemple, des constructeurs ou des méthodes plus pointues.
- Un joli exercice serait celui d'un objet animé comme un cercle tournant autour d'un autre à une vitesse régulière (la terre, la lune, les planètes).
- Ou encore un objet avec une certaine vitesse, une direction, qui gère les collisions avec les bords, et encore, plus difficile, qui gère les collisions avec les autres objets (fixes ou non, et pourquoi pas des objets images), transmet une partie de sa vitesse aux autres ou qui subit une déformation momentanée lors de la collision.

D'autres articles consacrés à BlueJ et écrit par l'auteur sont disponibles :

2ème partie - BlueJ et Netbeans	http://www.boichat.ch/javacpp/more/articleBlueJ_Netbeans.pdf
3ème partie - BlueJ et Greenfoot	http://www.boichat.ch/javacpp/more/articleBlueJ_Greenfoot.pdf
4ème partie - BlueJ et Eclipse	http://www.boichat.ch/javacpp/more/articleBlueJ_Eclipse.pdf