# IN201 : 1 – Classes et objets

Author : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

**Résumé**

Le but de ce TP est de manipuler les notions de classe et d'objet au travers de la classe `Point` vue en cours et de l'implantation de la classe `Orbite`.

## 1 Objectifs

Les objectifs de ce TP sont les suivants :
- se servir de façon simple de Subversion ;
- manipuler les outils du JDK avec les classes `Point` et `Orbite` ;
- comprendre l'utilité de la documentation javadoc à travers l'utilisation de la classe `Point` ;
- à partir de la conception de la classe `Orbite`, implanter cette classe ;
- comprendre les attributs et les méthodes et leur visibilité ;
- comprendre l'utilisation du constructeur ;
- comprendre la notion de référence ;
- documenter une classe ;
- aborder la problématique du test.

## 2 Préparation de l'utilisation de Subversion

Lors du cours, nous allons utiliser Subversion [1] pour gérer le code source de vos TP en version (cf. présentation qui vous a été faite de la problématique de la gestion de version). Chaque binôme possède son propre dépôt, qui n'est accessible que par lui-même. Ce dépôt est accessible depuis l'extérieur de l'ISAE.
L'URL de votre dépôt est pour cette semaine `https://eduforge.isae.fr/repos/IN201/0914/GPE/bNUM` où :
- `GPE` est le nom de votre groupe de PC en majuscule
- `NUM` est votre numéro de binôme (votre PC(wo)man les possède)

Par exemple, l'URL du dépôt du binôme 9 du groupe S est `https://eduforge.isae.fr/repos/IN201/0914/S/b9`. Pour connaître votre numéro de binôme, connectez-vous sur `http://www.tofgarion.net/lectures/IN201`.

Placez-vous dans le répertoire où vous mettrez tous vos TPs (créez-en un si besoin). Tapez ensuite la commande suivante

```
svn checkout URL .
```

où `URL` est l'adresse de votre dépôt (n'oubliez pas le « . » à la fin de la commande).
Votre répertoire est maintenant configuré pour interagir avec votre dépôt. Vous devriez y trouver un répertoire `TP1` contenant les fichiers et répertoires nécessaires pour ce TP.

Lors de votre première connexion, Subversion va vous poser une question à propos d'un problème de certificat. Choisissez l'option `p` pour accepter de façon permanente le certificat. De la même façon, Subversion vous demandera si vous souhaitez stocker votre mot de passe en clair dans un fichier. C'est vous qui voyez...
Si cela ne fonctionne pas, contactez votre PC(wo)man pour régler le problème avec le SI.

Si vous souhaitez ajouter un fichier dans votre archive, il faut auparavant que le répertoire le contenant appartienne déjà à l'archive. Pour cela, il suffit d'ajouter (`svn add`) le répertoire.
Attention, le comportement par défaut de Subversion lors de l'ajout d'un répertoire est d'ajouter également le contenu du répertoire (y compris les sous-répertoires). Pour éviter cela, utilisez `svn add -N` (*non-recursive*) lorsque vous voulez ajouter un répertoire.

Quelques remarques importantes :
- n'oubliez pas d'ajouter vos fichiers sources et de *commiter* vos changements au fur et à mesure de votre TP et quand vous l'aurez fini. Ne *commitez* pas trop souvent, faites-le quand vous avez fini d'implanter une classe par exemple. L'évaluation de votre TP ne se fera que sur les fichiers que vous aurez placés dans votre dépôt.
- n'ajoutez que vos fichiers sources dans le dépôt (pas les *bytecodes* en particulier !).
- maintenant que votre répertoire est géré en version, ne déplacez plus ou ne renommez plus des fichiers « à la main ». Subversion conserve des informations importantes dans un répertoire caché dans chacun de vos répertoires, si vous déplacez vos répertoires, ces informations ne seront plus à jour !

Le client Subversion en ligne de commande est normalement installé sur toute distribution Linux ou sur Mac OS X. S'il ne l'est pas, votre système de paquetages vous permettra de l'installer facilement. Sous Windows, vous pouvez utiliser TortoiseSVN [9], un client Subversion intégré à l'explorateur Windows. Sous Linux, RabbitVCS [7] fournit des fonctionnalités identiques.
Nous verrons à partir du prochain TP que nous utiliserons Subversion directement depuis l'environnement de développement Eclipse.

# 3 Conseils pratiques (à respecter !)

## 3.1 Organisation des répertoires

Chaque TP sera contenu dans un répertoire TP*n* où *n* est le numéro du TP. À l'intérieur de ce répertoire :
- le répertoire `src` contiendra vos sources Java
- le répertoire `classes` contiendra les *bytecodes* de vos classes et éventuellement les *bytecodes* des classes fournies pour le TP
- le répertoire `lib` contiendra les éventuels fichiers JAR nécessaires pour le TP
- le répertoire `doc` qui contiendra la documentation `Javadoc` de vos classes

Vous n'avez normalement pas à créer ces répertoires, ils seront créés lorsque vous mettrez à jour votre copie locale de votre dépôt Subversion au début du TP.

Si vous en avez besoin, pour créer un répertoire sous UNIX, la commande est `mkdir nomRepertoire`.

## 3.2 Édition des sources

Vous utiliserez pour éditer vos sources votre éditeur de texte habituel. Il est préférable d'un choisir un qui puisse indenter automatiquement votre code et le colorier syntaxiquement. Vous pouvez utiliser gedit [8] (commande `gedit` dans un shell) pour avoir un éditeur de texte minimal supportant la coloration syntaxique.
Geany [10] est un mini-environnement de développement plus évolué que gedit que vous pouvez utiliser au CI.
Si vous utilisez Emacs [3], vous pouvez utiliser JDEE [4] pour avoir un environnement de développement complet. Attention, n'utilisez pas Emacs si vous ne le connaissez pas.
Vous avez à votre disposition sur le site des conventions de code. Respectez les. En particulier :
- les noms de classes commencent par une majuscule ;
- les noms d'attributs et de méthodes commencent par une minuscule ;
- les noms de variables commencent par une minuscule ;

- en cas de nom composé, on ne sépare pas les mots, mais on commence chaque mot par une majuscule (ex : `methodeCalcul`) ;
- on place un espace avant et après un opérande.

### 3.3 Compilation et exécution

Les deux opérations principales que vous effectuerez sont la compilation des sources et l'« exécution » de vos programmes. Il est donc préférable d'avoir deux terminaux ouverts, un pour la compilation, un autre pour l'exécution.
Le terminal servant à la compilation sera positionné dans le répertoire `src`. Pour compiler une classe Java, on utilisera la commande suivante :

```
javac -d ../classes -cp .:../classes Classe.java
```

L'option `-d ../classes` signale au compilateur de placer les fichiers *bytecode* dans le répertoire `classes` que vous aurez créé précédemment.
L'option `-cp .:../classes` signale au compilateur que les fichiers *bytecode* dont il peut avoir besoin sont soit dans le répertoire courant, soit dans le répertoire `classes` précédemment créé (nous reparlerons de cette notion la prochaine séance).
Le terminal servant à l'exécution sera positionné dans le répertoire `classes`. La commande d'interprétation d'une classe sera :

```
java Classe
```

où `Classe` est le nom de la classe à interpréter.

### 3.4 Génération de la documentation

Pour générer la documentation de vos classes via l'utilitaire `javadoc`, il suffit de vous placer dans le répertoire `src` et d'utiliser la commande :

```
javadoc -d ../doc -version -author *.java
```

L'option `-d ../doc` signale à `Javadoc` de placer la documentation dans le répertoire `doc`.

### 3.5 Organisation de votre espace de travail

Organisez votre espace de travail afin de ne pas avoir des fenêtres superposées et utilisez les bureaux à votre disposition. Par exemple :

- bureau 1 : éditeur de texte
- bureau 2 : les deux terminaux (compilation et exécution) côte à côte
- bureau 3 : navigateur web pour la documentation

## 4 Manipulation de la classe `Point`

L'objectif de cette section est de manipuler la classe `Point` déjà vue en cours. On ne dispose que du *bytecode* de `Point` (dans le répertoire `classes` de votre TP) et de sa documentation Javadoc sur le site du cours. Le code source de la classe sera mis à votre disposition sur le site lors de la publication du corrigé de ce TP.

1. créer une classe `TestPoint` pouvant être interprétée. La compiler et vérifier qu'elle peut être interprétée.
2. créer un nouveau point de coordonnées $(2, 3)$ affecté à une référence `p` et l'afficher.
3. translater le point référencé par `p` d'un vecteur $(-5, 3)$ et afficher le point.
4. déclarer une nouvelle référence `q` de type Point et lui affecter la référence `p`. Translater l'objet référencé par `q` d'un vecteur $(1, 1)$ et afficher les objets référencés par `p` et `q`. Que se passe-t-il ? Pourquoi ?
5. utiliser maintenant une méthode disponible dans l'API [1] de `Point` pour copier un point et affecter le point copié à une référence `r`. Afficher les objets référencés par `p` et `r`. Translater l'objet référencé par `r` d'un vecteur $(2, 2)$ et afficher les objets référencés par `p` et `r`. Que se passe-t-il ? Pourquoi ?
6. copier de nouveau `p` dans la référence `r`. En utilisant l'opérateur ==, afficher le résultat de la comparaison entre `p` et `q` et entre `p` et `r`. Que se passe-t-il ? Pourquoi ?

1. API : *Application Programming Interface*, les services pouvant être utilisés depuis l'extérieur d'un module, d'une classe etc. Dans le monde Java, l'API d'une classe est documentée via Javadoc.

7. trouver dans l'API de `Point` une méthode permettant de comparer *logiquement* (i.e. en utilisant l'état des objets) deux points et refaire les comparaisons précédentes. Que se passe-t-il ? Pourquoi ?

N'oubliez pas de *commiter* le code source de votre classe `TestPoint`.

## 5 Implantation d'une classe `Orbite` pour vol en formation

Les notions présentées lors de ce TP sont simplifiées par rapport à la réalité afin de faciliter l'implantation de la classe. Vous vous référerez au cours IS201 « Mécanique spatiale » [2] pour avoir une vision plus rigoureuse du problème.

Nous rappelons le problème vu en cours : on cherche à modéliser une orbite elliptique autour de la Terre, orbite sur laquelle trois satellites en formation sont positionnés avec un angle constant de $\frac{2\pi}{3}$. Chaque satellite possède un instrument avec une ouverture de $\frac{\pi}{4}$ permettant de communiquer avec le satellite le précédant (il suffit que le satellite soit dans le cône de visibilité de l'instrument). L'instrument est toujours positionné à $\frac{\pi}{4}$ par rapport à la tangente de la trajectoire du satellite grâce à un asservissement. Le problème est présenté sur la figure 1.

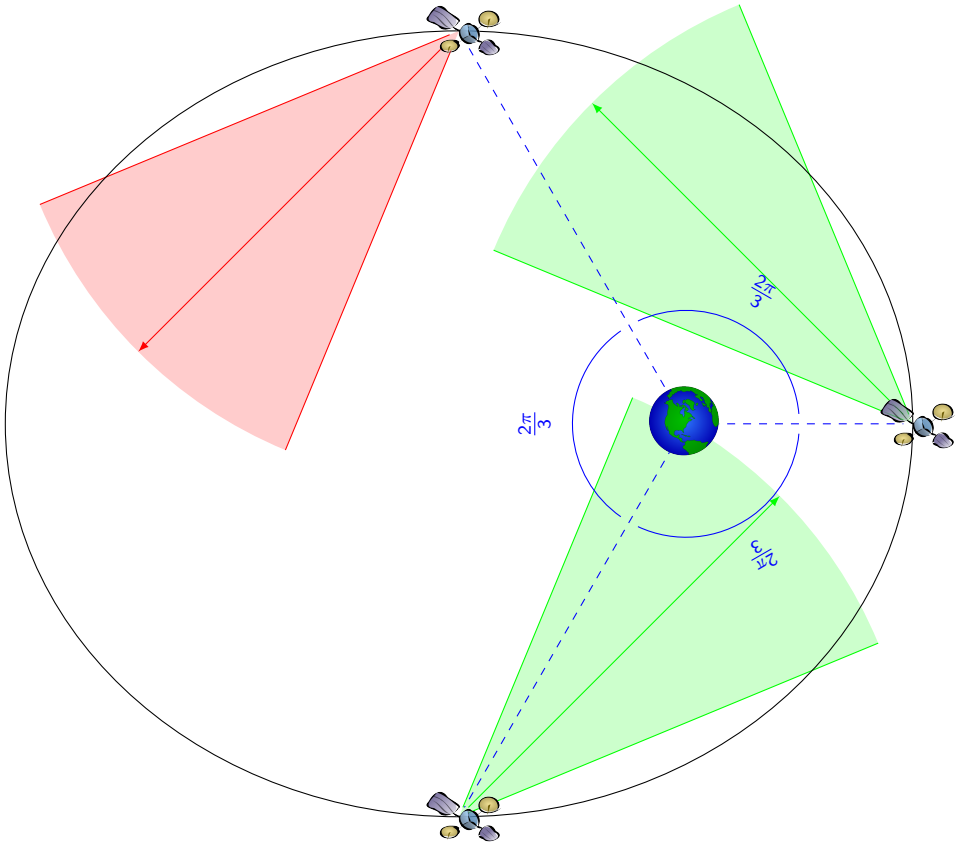


FIGURE 1 – L'orbite à développer et ses 3 satellites. Seul l'instrument du satellite situé à $\frac{2\pi}{3}$ est présenté ici.

On pourra prendre comme cas de tests les deux orbites réelles suivantes :

| | $e$ | $a$ |
|---|---|---|
| SPOT | 0,0 | $26 \cdot 10^6$ |
| EXOSAT | 0,93 | $100 \cdot 10^6$ |

On rappelle que si l'on définit une ellipse par son excentricité $e$, son demi-grand axe $a$ et son centre de coordonnées $(x_c, y_c)$ (cf. figure 2), on a les relations suivantes (vous verrez que $v$ est appelé *anomalie vraie* et $\theta$ *anomalie excentrique*) :

| | |
|---|---|
| dimensions | $b = a\sqrt{1-e^2}$<br>$c = ae$ |
| équation paramétrique | $x = x_c + a\cos\theta$<br>$y = y_c + b\sin\theta$ |
| relation entre $\theta$ et $v$ | $\tan\theta = \dfrac{\sqrt{1-e^2}\sin v}{e+\cos v}$ |
| équation polaire | $r = \dfrac{a(1-e^2)}{1+e\cos v}$ |
| tangente en $(x_0, y_0)$ | $\dfrac{(x-x_c)(x_0-x_c)}{a^2} + \dfrac{(y-y_c)(y_0-y_c)}{b^2} = 1$ |

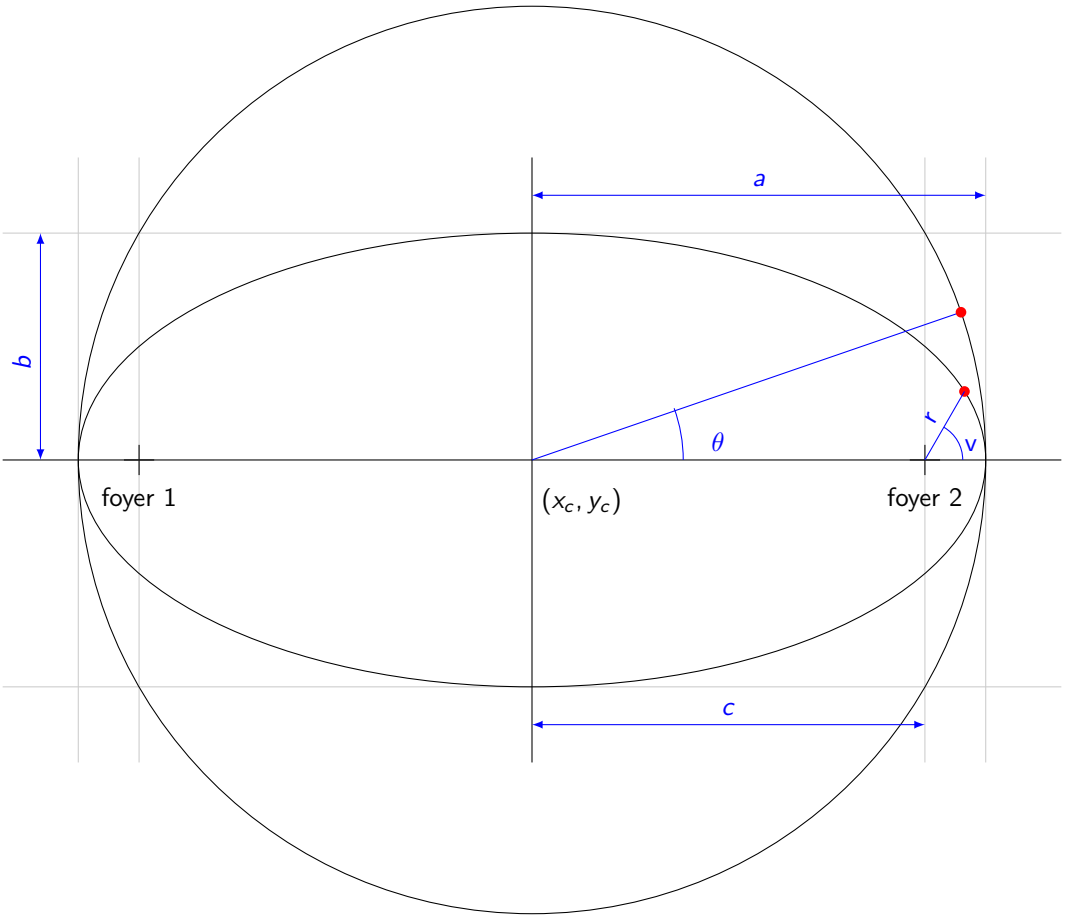


FIGURE 2 – Ellipse et ses grandeurs

Le diagramme de classe d'implémentation de la classe `Orbite` à utiliser est présenté sur la figure 3. Quelques remarques :

- les trois premiers attributs sont des attributs *statiques* représentant des constantes.
- les attributs nous montrent que l'état d'une orbite est caractérisé par $e$, $a$, $b$ et la position du foyer représentée par un point.
- `Orbite` possède deux constructeurs. Tous deux prennent comme paramètres $e$ et $a$, le premier prenant en plus un point représentant le foyer en paramètre. On en déduit que :
  - le deuxième constructeur initialise le foyer avec le point $(0, 0)$
  - $b$ est un attribut *dérivé* calculable à partir de $e$ et $a$. La méthode *privée* `calculerB` permet de faire ce calcul
- $c$ est une valeur caractéristique de l'ellipse, mais elle n'est pas stockée comme attribut. Elle possède toutefois un accesseur
- pour calculer les coordonnées d'un point sur l'ellipse, on dispose de deux méthodes :

  - `calculerPointSurOrbite` qui prend en paramètre $\theta$, i.e. l'anomalie excentrique
  - `calculerPointSurOrbiteFoyer` qui prend en paramètre $v$, i.e. l'angle par rapport au foyer de l'ellipse (anomalie vraie). Cette méthode sera implantée à l'aide de la méthode privée `calculerTheta` qui permet de calculer $\theta$ à partir de $v$.
- `calculerVecteurTangent` permet de calculer un vecteur tangent à l'ellipse en un point situé par $\theta$. La composante en $x$ de ce vecteur doit aller « dans le sens » de la vitesse du satellite.
- `voitSatelliteSuivant` permet de vérifier si un satellite voit le satellite situé à $\frac{2\pi}{3}$ de lui par rapport au foyer. La position du satellite est donnée par $v$.

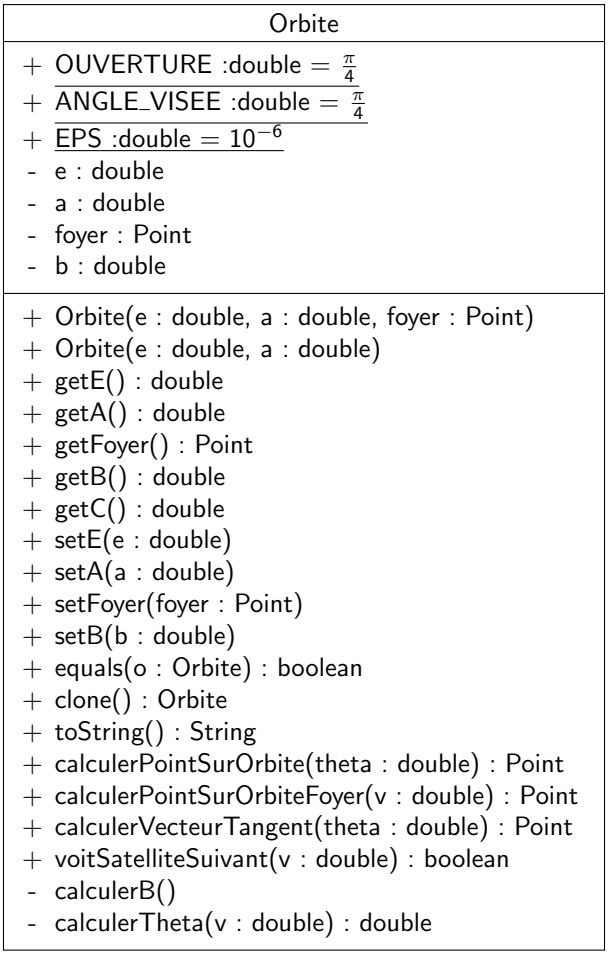


FIGURE 3 – Diagramme de classe d'implantation de la classe `Orbite`

Nous proposons de développer la classe `Orbite` de façon itérative en suivant les questions qui suivent. Vous écrirez en même temps une classe applicative `TestOrbite`. À chaque fois que vous développez une méthode :

- écrivez la documentation de la méthode
- testez-là via votre classe `TestOrbite`

1. à partir du diagramme UML, écrire le squelette de `Orbite` et vérifier qu'il compile.
2. écrire les constructeurs de `Orbite`, ainsi que les accesseurs et modifieurs définis dans le diagramme UML. Que se passe-t-il lorsque l'on translate le point ayant servi à construire l'ellipse ? Est-ce correct ?
3. écrire les méthodes `clone`, `equals` et `toString`. Pour la méthode `toString`, on utilisera les paramètres de l'ellipse et les coordonnées de son foyer par exemple.
4. écrire la méthode permettant de calculer un point à partir de $\theta$. On utilisera la documentation de la classe `Math` (cf. [6]) pour trouver les fonctions trigonométriques et autres nécessaires.
5. écrire la méthode permettant de calculer un point à partir de $v$.

   Vous pouvez utiliser à partir de maintenant une petite interface graphique développée en Swing, la bibliothèque d'IHM fournie avec le JDK [5], pour visualiser le déplacement des satellites sur vos orbites. Nous ferons une séance sur Swing vous expliquant comment développer de telles IHM.

   Pour pouvoir utiliser l'IHM, il vous suffit dans votre répertoire `classes` d'exécuter la commande suivante :

```
java -cp .:../lib/lab1-gui.jar LancerGUI
```

Les *bytecodes* de l'IHM sont contenus dans une archive JAR (*Java ARchive*) `lab1-gui.jar` située dans le répertoire `lib`. L'option `-cp .:../lib/lab1-gui.jar` permet de préciser à `java` de chercher les *bytecodes* nécessaires soit dans le répertoire courant, soit dans l'archive.

La visualisation graphique de vos orbites peut vous aider mais ne consiste pas un test à part entière...

6. écrire la méthode permettant de trouver le vecteur tangent en un point de l'ellipse. On utilisera la classe `Point` pour représenter un vecteur. Pour simplifier les calculs, on se raménera à une ellipse dont le centre est à l'origine. On fera attention au signe de l'angle à l'origine et aux cas limites.

L'opérateur `%` de Java permet de calculer le modulo entre deux nombres et fonctionne avec les réels. Cela peut vous servir.

7. implanter la méthode `voitSatelliteSuivant`. Quelques indications :
   - sur l'orbite GPS les satellites sont toujours visibles ;
   - sur l'orbite EXOSAT, on peut utiliser le tableau suivant pour faire les tests

| $v$ | visibilité |
|---|---|
| $[0.000; 1.246]$ | oui |
| $[1.247; 3.169]$ | non |
| $[3.170; 3.348]$ | oui |
| $[3.349; 4.276]$ | non |
| $[4.277; 2\pi]$ | oui |

N'oubliez pas de *commiter* le code source de vos classes.

## Références


[1] Apache Foundation. *Apache Subversion*. 2013. URL : http://subversion.apache.org/.

[2] Bénédicte Escudier et Jean-Yves Pouillard. *Mécanique spatiale*. Notes de cours du module IS201. ISAE, 2009. ISBN : 978-2-84088-242-8.

[3] Free Software Foundation. *GNU Emacs*. 2013. URL : http://www.gnu.org/software/emacs/.

[4] Paul Kinnucan. *JDEE – a Java development environment for Emacs*. 2013. URL : http://jdee.sourceforge.net/.

[5] Oracle. *Creating a GUI With JFC/Swing*. 2013. URL : http://docs.oracle.com/javase/tutorial/uiswing/.

[6] Oracle. *Java API specifications*. 2013. URL : http://docs.oracle.com/javase/7/docs/api/index.html.

[7] Adam Plumb et Jason Heeris. *RabbitVCS*. 2013. URL : http://rabbitvcs.org/.

[8] The Gnome project. *gedit text editor*. 2013. URL : http://projects.gnome.org/gedit/.

[9] The TortoiseSVN development team. *TortoiseSVN*. 2013. URL : http://tortoisesvn.net/.

[10] Enrico Tröger et al. *Geany*. 2013. URL : http://www.geany.org/.


## License CC BY-NC-SA 3.0