# Devoir de programmation objet
# Licence d'Informatique & ILOG1

Marc Champesme
Marc.Champesme@lipn.univ-paris13.fr
Departement d'Informatique
Institut Galilée

20 décembre 2002

## 1 Objectif du devoir

Le but est de mettre en pratique les concepts de la programmation orientée objet et plus particulièrement :
- bien structurer le code d'une application en classes et méthodes ;
- écrire et commenter un programme afin qu'il puisse être réutilisé ;
- utiliser correctement l'héritage et le polymorphisme ;
- utiliser la programmation par contrat.

## 2 Présentation du sujet

Il s'agit de réaliser un tableur très simple. Pour réduire la quantité de travail nécessaire, le code source en JAVA d'une version préliminaire du logiciel vous est fourni (disponible à l'adresse `http://marc.champesme.free.fr/POO/source/tableur.tar.gz`). Cependant, cette version du logiciel possède un certain nombre de défauts :
- le code source ne possède aucun commentaire ;
- ce programme n'utilise pas les concepts de la programmation par contrat : il ne possède aucune assertion ;
- tous les identificateurs présents sont en anglais ;
- aucune séparation entre le code gérant l'interface et le reste du code ;
- quelques abus dans l'utilisation des variables de classes ;
- la répartition du code est mal faite (i.e. les classes ne sont pas correctement définies) : certaines méthodes ou attributs ne sont pas à leur place ;
- il manque certaines fonctionnalités.

## 3 Travail demandé

Le devoir doit être réalisé individuellement : chaque étudiant devra rendre sa propre version du logiciel sur laquelle il sera évalué. Le travail à réaliser consiste à :
- Modifier le code de l'application fournie, afin qu'il satisfasse les standards de programmations exposés à la fin de ce document ;
- Ajouter certaines fonctionnalités au logiciel.

### 3.1 Corrections à apporter au logiciel

Afin de rendre la gestion de l'interface indépendante de la gestion des données du tableur, les caractéristiques (attributs ou méthodes) liées à l'affichage doivent être placées dans des classes distinctes de celles assurant le traitement des données. Par exemple, dans la classe `Cell`, les méthodes assurant la modification des données (`setValue()` et `setRawValue()`) restent dans la classe `Cell`, tandis que les méthodes assurant l'interaction avec l'utilisateur (affichage et récupérations des actions de l'utilisateur

comme les clicks de souris) doivent être déplacées dans les classes assurant l'affichage (de la cellule ou du tableur).

Faire des classes distinctes pour les trois types de cellules (avec utilisation de l'héritage et du polymorphisme).

Mettre dans une classe séparée de la classe `Cell` tout ce qui concerne l'analyse syntaxique des formules.

### 3.2 Fonctionnalités à ajouter

- possibilité d'insertion de lignes et de colonnes (avec mise à jour automatique des formules) ;
- gestion des déplacements avec les flèches du clavier.

## 4 Evaluation

Compte tenu de l'objectif fixé, ce n'est pas la richesse en fonctionnalité, mais plutôt la qualité de la réalisation qui sera évaluée.

La qualité de la réalisation sera évaluée selon les critères suivants :

- architecture de l'application : bonne répartition du code en classes, bon usage de l'héritage, bonne séparation entre le code gérant l'interface utilisateur et le reste du code...
- conception et réalisation favorisant la réutilisabilité du code ;
- utilisation adéquate de la programmation par contrat (dans la définition des assertions *et* l'implémentation des méthodes) ;
- bonne documentation des classes ;
- fonctionnement de l'application : ergonomie (facilité d'utilisation), absence de bogue (y compris lorsque l'exécution se fait avec contrôle des assertions activé).

## 5 Documents à rendre par chaque étudiant

Chaque étudiant rend une disquette et un rapport sur papier contenant :

- Description des modifications apportées à l'application initiale (fichier sur la disquette + version imprimée) ;
- La documentation (produite par `javadoc`) des classes (fichiers html sur la disquette) ;
- Le code de toute l'application et le fichier `build.xml` permettant l'utilisation avec Jass (sur la disquette).

## 6 Standards de programmation

**Commentaires** Les commentaires doivent être rédigés en respectant les règles suivantes :

- Tous les commentaires doivent être en français ;
- Chaque classe, méthode ou champ public doit posséder son propre commentaire au format javadoc.

**Choix des identificateurs** Tous les identificateurs doivent être francisés. Les seules exceptions admises sont les préfixes `get` et `set` pour les méthodes servant à consulter ou modifier un champ particulier : par exemple, il est admit que l'identificateur de méthode `setCenter` soit traduit (de manière incomplète) en `setCentre`. Pour les attributs privés (visibilité `private`), les identificateurs doivent avoir le préfixe '_' (exemple : `_centre`).

**Assertions** Vous devez définir un invariant pour chaque classe et, sauf cas particulier (qui doit rester rare), toute méthode doit posséder une pré-condition et une post-condition.

Vous devez vous efforcer d'écrire les assertions en JAVA de façon à ce qu'elles puissent être activées à l'exécution. Cependant certaines assertions ne peuvent être décrites de manière satisfaisante en JAVA, dans ce cas, les assertions pourront être écrites en français.

**Visibilité / masquage d'information** Toutes les classes et les méthodes doivent être déclarées `public` et tous les attributs doivent être déclarés `private`. Il est néanmoins admis que les attributs `final` soient déclarés `public`.

Toute exception à cette règle devra être clairement justifiée dans les commentaires.

**Variables de classe** L'usage de variables de classes devra être réservé à des cas exceptionnels (par exemple variables `final`).