TP 1

L'objectif de ce premier TP est une première prise de contact avec les objets et certains des concepts qui y sont liés (classe, instance, envoi de messages, etc.). Vous découvrirez les premiers éléments de syntaxe JAVA.
Les différentes manipulations se feront à travers l'environnement BLUEJ[1]. Nous n'utiliserons cet environnement que lors des 2 ou 3 premières séances de TP, le temps pour vous d'en apprendre plus sur l'écriture de code Java. L'objectif n'est pas de faire toutes les manipulations le plus vite possible ! Vous devez comprendre ce que vous faites et interpeler votre enseignant dès qu'il y a quel que chose que vous ne comprenez pas ou n'êtes pas sûr de bien comprendre.

Démarrage : configuration de java et bluej Nous utiliserons la version du jdk1.6 dans l'unité POO. PPour pouvoir utiliser Java avec votre compte, il vous faut modifier le fichier `.bashrc` à la racine de votre espace de fichiers[2] en y ajoutant les lignes :

```
java_version=1.6.0_26
java_path=/opt/java/jdk${java_version}/bin
jre_path=/opt/java/jdk${java_version}/jre/bin
PATH="${java_path}:${jre_path}:$PATH"
```

La documentation se trouve à l'url suivante à ajouter dès maintenant à vos marque-pages

`/opt/java/jdk1.6.0_26/docs/api/index.html`

Mise en place de l'espace de travail :

- Créez un répertoire `po/tp1` (par exemple, via la console) :
  1. depuis la racine de votre compte : "`mkdir po`" puis "`cd po`" puis "`mkdir tp1`" puis "`cd tp1`"
  2. Récupérez sur le portail le fichier `fichiers-tp1.tar.gz` "qui va avec" ce TP et placez le dans votre répertoire tp1.
  3. Allez dans le répertoire `tp1` créé ci-dessus et décompressez cette archive :
     `tar xvzf fichiers-tp1.tar.gz`

  ou utilisez le navigateur de fichiers.
- Ouvrez un terminal et lancez BLUEJ[3] : `bluej &`

Exercice 1 : Prise en main
Dans BLUEJ : Project → Open Project... puis ouvir le projet `Tv` du répertoire `po/tp1`.
Dans la fenêtre de projet apparaissent différentes icônes :

- en haut à gauche, une icône désignant le fichier `README.TXT` permettant de donner une description du projet. Un double-click permet d'afficher ce fichier et de le modifier si besoin.
- au centre, une icône saumon de titre Tv. Elle représente la classe de nom `Tv`. Un double-click permet d'afficher le source de la classe ou la documentation (appelée "JavaDoc"), on peut passer d'un affichage à l'autre par l'intermédiaire de la liste déroulante en haut à droite. Choisissez Source Code qui correspond au code Java.

Dans la fenêtre projet, activez le menu View → Show Terminal puis dans la fenêtre qui s'ouvre : Options → Record Method Calls
Un rapide coup d'œil sur le source permet d'avoir un premier aperçu d'un code JAVA. Les commentaires font apparaître trois blocs :

les attributs ils définissent l'état des instance (objets) de la classe, càd les propriétés qui les caractérisent

le constructeur permet de construire les objets conformes au modèle définit par la classe

les méthodes définissent les traitements que l'on peut *invoquer* sur une instance de la classe.

[1] Cet environnement est récupérable sur le site `www.bluej.org`, il est gratuit. De même le langage Java est disponible depuis le site de oracle : `java.oracle.com`.
[2] Sous linux, les fichiers dont les noms commencent pas un "`.`" sonrt des fichiers cachés.
[3] Pour éviter des problèmes avec Bluej il est prudent de désactiver les effets visuelsdu bureau. Aller dans le menu Système→Préférences→Apparence et dans l'onglet Effs Visuels cjhsoir Aucun

Les valeurs des attributs peuvent varier d'une instance à une autre, le code des méthodes est le même pour les instances, même si l'effet du traitement peut varier parce qu'il utilise (le plus souvent) les attributs.
Sans rentrer dans le détail de ce code maintenant, les différents points s'éclairciront au cours des prochaines séances, jetez un œil sur le cde pour vous familiariser avec la syntaxe java, il n'y a rien de très compliqué à deviner.
Intéressons nous au menu déroulant en haut à droite de la fenêtre, le choix actuel est Source Code. Activez l'autre choix : Documentation.
Le contenu de la fenêtre change alors et vous propose la documentation sur la classe `Tv`. La structure de cette documentation est commune à tous les projets JAVA. Elle est générée à partir de commentaires insérés dans le source (en y jetant à nouveau un œil, vous devriez pouvoir deviner lesquels) et de l'outil JAVADOC que nous découvrirons prochainement.
Cette documentation est un document hypertexte qui présente les différents points suivants :

1. le nom de la classe
2. une description de la classe et des informations "annexes" (auteur, version, etc.)
3. le(s) constructeur(s)
4. la signature des méthodes (avec hyperlien vers les détails)
5. les détails des méthodes. Par exemple pour la méthodes `changeChaine` vous avez un descriptif du paramètre (rubrique `Parameters`) et pour la méthode `chaineCourante` un descriptif du résultat (rubrique `Returns`).

Il faudra s'habituer à la lecture de documentation dans ce format.

Exercice 2 : Première classe, premières manipulations
Au fur et à mesure des manipulations, veillez à consulter le contenu de la fenêtre "Terminal Window" afin de visionner la syntaxe des invocations JAVA (si vous avez bien activé l'option "Record Method Calls" comme demandé précédemment) et leurs résultats. Vous pouvez réinitialiser le contenu de cette fenêtre par `Option` → `Clear`.

Revenons à la fenêtre projet, contenant la classe `Tv`.

compilation un click droit sur l'icône de la classe ouvre un menu contextuel : choisissez Compile pour compiler la classe (le dessin de l'icône change). L'icône de classe change de couleur pendant sa compilation.

création d'instance pour l'exploitation de la classe, il est nécessaire de créer au moins une instance. Une classe définit une structure, ou modèle, et les instances d'une classe sont les objets qui obéissent à ce modèle. Choisir `new Tv()` dans le menu contextuel et choisissez un identificateur/référence pour votre instance (par défaut `tv1` est proposé). Une représentation de l'objet apparaît dans la zone grisée située en partie inférieure de la fenêtre. La syntaxe de la création de l'instance apparaît dans la fenêtre du terminal (`new Tv();`).

visualisation de l'état click droit sur l'instance puis choisissez *Inspect* ; un double-click produit le même résultat. Dans la fenêtre qui apparaît on visualise les différents attributs qui définissent l'état de notre objet ainsi que leurs valeurs.

envoi de message/invocation de méthode activez le menu contextuel sur l'icône de l'objet (clic droit). On voit apparaître la liste des comportements, les *méthodes*, disponibles pour l'objet. Cette liste est définie par le type de l'objet c'est-à-dire sa *classe*. Choisissez `chaineCourante()`. Une fenêtre s'affiche vous donnant le résultat retourné par la méthode dont vous pouvez vérifier la signature et le descriptif dans la documentation.
Dans le Terminal vous pouvez voir le code java correspondant à cet appel de méthode.Notez l'usage du ".".
Vous pouvez également consulter le "source code" de cette méthode pour comprendre le traitement qui a été exécuté et qui est très simple dans ce cas.

encore Invoquez la méthode `allume()`, vous pouvez visualiser dans la fenêtre d'inspection le changement de d'état de l'objet. La valeur de son attribut `allume` est passée de `false` à `true`. Etudiez également le code.

et encore Invoquez la méthode `afficheEtat()`, cette fenêtre provoque un effet de bord en affichage et un terminal s'ouvre donc pour en permettre la visualisation (sauf si vous aviez déjà ouvert ce terminal comme cela était demandé). A nouveau jetez un œil au code.

Exercice 3 : Première classe, autres manipulations
Gardez toujours la fenêtre du terminal active. Consultez-la régulièrement au cours des manipulations suivantes afin de vous familiariser avec la syntaxe JAVA associée aux différentes invocations.

instances Créez une novuell instance de la classe `Tv` et visualisez son état.

invocations Effectuez quelques invocations sur cette instance créée, consultez la documentation pour un bon usage des méthodes. Certaines méthodes requièrent que l'on fournisse un argument, le type de cet argument est alors rappelée dans la fenêtre qui s'ouvre.

Etudiez simultanément le code des méthodes invoquées afin de vous familiariser avec la syntaxe.

À voir dans le source :

- le constructeur, la déclaration d'attributs, la déclaration de méthodes, avec ou sans argument, avec ou sans valeur de retour et l'utilisation du `return`,
- la délimitation de bloc par les accolades { ... },
- la syntaxe de la structure conditionnelle `if`, avec ou sans `else`
- les commentaires, "`/* .. */`" et "`// ...`",
- l'opérateur `+` de concaténation des chaînes, et les chaînes notées entre guillemets doubles `"`.
- le `System.out.println` qui permet un affichage.

Exercice 4 : Seconde classe
(Conservez par la suite toujours la fenêtre du terminal ouverte afin de visualisez les invocations JAVA réalisées)

Ouvrez le projet `Livre1` (vous pouvez fermer `Tv`). Deux classes appartiennent cette fois à ce projet. Vous pouvez remarquer que les icônes des deux classes sont reliées par une flèche. Celle-ci indique qu'il existe une relation de dépendance entre les classes : la classe `Bibliotheque` utilise des instances de la classe `Livre`. Cette information est importante. Elle souligne que toute modification de la classe `Livre` peut avoir des répercussions sur le fonctionnement de la classe `Bibliotheque` et d'autant plus si l'on modifie l'interface publique[4] de `Livre`.

Q 1 . Créez une instance de `Bibliotheque`. Si vous n'y arrivez pas, peut être avez-vous oublié une étape ? Référez vous à l'exercice 2.

Q 2 . Créez deux instances de `Livre`. Trois des arguments du constructeur sont des objets de classe `String`. Cette classe est un peu particulière et vous pouvez utiliser comme valeur d'instance de cette classe toutes les constantes chaînes de caractères (classe `String`) qui se notent **entre guillemets**. Par exemple, vous pouvez créer un objet par :

```
new Livre("JRR Tolkien","Le Seigneur des Anneaux",1954,"...")
```

Q 3 . Examinez les états de ces deux instances de `Livre`.

Q 4 . Invoquez la méthode `ajouteLivre` sur l'objet `Bibliotheque` créé. Cette méthode prend pour argument une instance de la classe `Livre`, vous pouvez donc donner comme argument l'identifiant de l'une des deux références d'instance dont vous disposez (`livre1` par exemple). Attention, il s'agit ici d'une référence et non pas d'une chaîne de caractères, et donc pas de guillemets !

Q 5 . Effectuez diverses manipulations avec ce projet.

À voir dans le source :

- un constructeur avec paramètre dans `Livre`,
- l'utilisation de `this` dans le constructeur de `Livre`,
- le modificateur `private` sur les attributs, et `public` sur les méthodes,
- la syntaxe de la structure itérative `while`,

Exercice 5 : Encore des Livres
Ouvrez le projet `Livre2`. Vous retrouvez les élements précédents avec en plus une classe `Auteur`. La structure des instances de la classe `Livre` a changé et est maintenant construite à partir d'une instance de la classe `Auteur` (ce qui explique la présence de la flèche entre les deux icônes de classe).

Q 1 . Petit retour sur la compilation :

Q 1.1. Examinez le contenu de votre répertoire "*chez-vous*`/tp1/livre2`". Il contient les fichiers source d'extension `.java` correspondant au code des trois classes.

Q 1.2. Passez au premier plan la fenêtre du projet `livre2`, sélectionnez la classe `Bibliotheque`, puis dans le menu contextuel choisissez Compile. On remarque, au changement de couleur de leurs icônes, que les classes `Auteur` et `Livre` sont (dans cet ordre) également compilées.

Le compilateur prend en compte les dépendances entre classes, et, lors de la compilation d'une classe, les autres classes non encore compilées dont elle dépend sont également compilées (et donc récursivement...).

[4] Les envois de message autorisés

Q 1.3. On peut à nouveau examiner le contenu du répertoire *chez-vous*`/tp1/livre2` et constater l'appaition de fichiers d'extension `.class`. Il s'agit du *bytecode* généré par le compilateur JAVA pour chacune des classes compilées. Vous pouvez par curiosité ouvrir ce fichier `.class` dans un éditeur de texte, vous pourrez constater qu'il n'est pas totalement incompréhensible mais pas lisible pour autant.

Q 1.4. Modifiez le source de la classe `Auteur` (ajoutez un espace n'importe où par exemple et sauvez). Dans la fenêtre projet, l'icône de `Auteur` signale la modification (hachures), mais également, l'icône de le classe `Livre`. Cela traduit en fait que dans la mesure où une classe dont dépend la classe `Livre` a été modifiée (`Auteur` en l'occurrence), il est possible que cela ait des répercutions sur la classe `Livre`. BLUEJ impose ici une recompilation pour contrôler l'impact ou non de ces répercussions.

Q 2 . Créez des instances des différentes classes (il y a un ordre à respecter...).
Attention, cette fois le premier paramètre du constructeur pour un `Livre` prend en paramètre un objet `Auteur`, pas une chaîne de caractères. Il faut donc indiquer une référence vers un objet `Auteur`, qui doit donc avoir été préalablement créé.

Q 3 . Inspectez une instance de `Livre`. Sélectionnez dans la fenêtre, la ligne correspondant à l'attribut `auteur`, le bouton Inspect sur la droite de la fenêtre s'active, en cliquant dessus vous pouvez examiner l'état de cet attribut.

Q 4 . Effectuez différentes manipulations sur ce projet.

À voir dans le source :

- le champ `auteur` de la classe `Livre` est un objet de la classe `Auteur`,
- la méthode `getAuteur()` retourne une valeur objet,
- l'utilisation de la notation "`.`" pour invoquer une méthode sur un objet (`auteur.affiche()`),

Exercice 6 : Robbie le robot
Ouvrez le projet `Robot`.

Q 1 . Créez trois tapis roulants acceptant des poids maximum différents (10, 100 et 300 par exemple).

Q 2 . Créez des instances de caisses (de différents poids) et `Robbie` le robot.

Q 3 . Faites déposer par `Robbie` les caisses sur les tapis et autres invocations de méthodes[5].

À voir dans le source :

- l'opérateur de négation "`!`" (= *not*), la valeur `null`, la syntaxe de la structure itérative `for`, l'indentation,
- le `import` en début de code de `TapisRoulant`, qui indique que l'on souhaite utiliser une classe définie dans un autre paquetage, `java.util.*` ici, et l'utilisation de la collection `ArrayList` (définie dans le paquetage `java.util`), l'ensemble sera détaillé utltérieurement.

Exercice 7 : Usine à jouets
Ouvrez le projet `jouets`. En invoquant la méthode `fabrique()` sur une instance de `Usine` vous pouvez construire une instance de `Jouet`, vous pouvez "récupérer" et nommer l'instance créer en cliquant que le bouton Get de la fenêtre qui affiche le résultat.
À voir dans le source :

- il peut y avoir plusieurs constructeurs pour une même classe,
- un objet créé "par un autre objet" lors de l'invocation d'une méthode, utilisation du `new` et un objet peut être résultat d'une méthode.

Ce qu'a mis en évidence ce tp

- **Écrire le programme c'est "coder" des classes, l'exécuter c'est manipuler des objets** (qui sont instances des classes codées).
- **L'exploitation se fait par l'intermédiaire d'instances créées en cours d'exécution à partir des classes définies en phase de conception.**
- **Toute manipulation se fait à travers un objet en lui envoyant un message. Les messages possibles sont définis par la classe. L'envoi de message se fait en utilisant une référence vers l'objet.**

[5] Rien ne vous empêche ici (à tort) de placer plusieurs fois la même caisse, y compris sur des tapis différents ! Mais pour l'instant, l'important est de manipuler...