

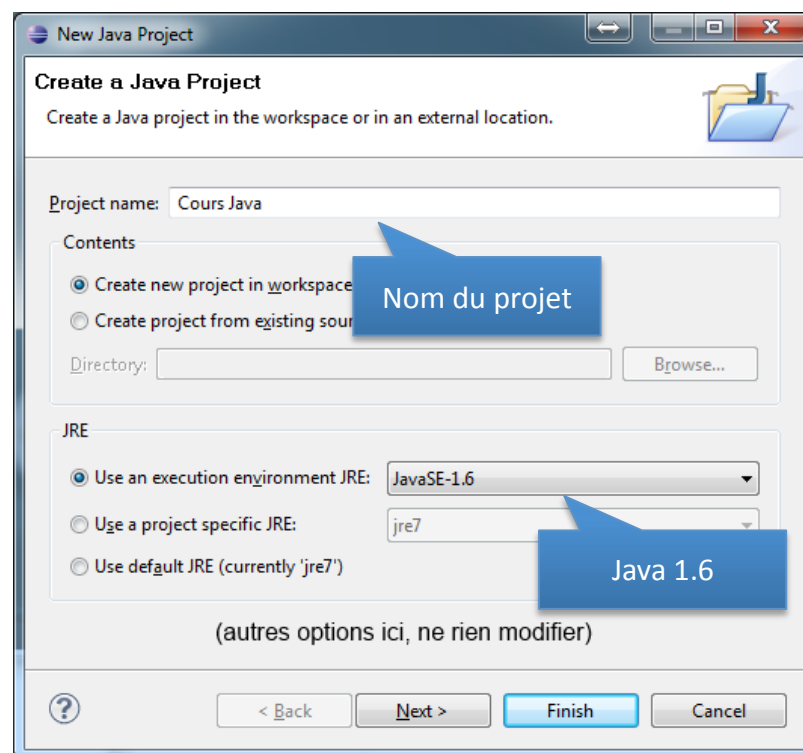
TP 1-1 : Premiers objets en Java

Objectifs : Dans ce TP, nous verrons comment créer un programme en Java avec Eclipse et passerons en revue les concepts fondamentaux de la programmation orientée objet.

Exercice 1 : Hello, World !

Cet exercice a pour objectif de vous enseigner comment créer un projet, une classe et une méthode principale dans votre programme, en utilisant Eclipse.


1. Ouvrir Eclipse. S'il vous est demandé quel « workspace » choisir, cliquez simplement sur « OK ». Le workspace Eclipse est l'endroit où le logiciel enregistre vos projets.
2. Si un écran de bienvenue vous est présenté, fermez l'onglet « Welcome », en haut à gauche. Vous devriez à présent vous trouver devant l'interface principale d'Eclipse
3. Cliquez sur File -> New -> Java Project. Remplissez le menu comme suit puis cliquez sur « Finish » :

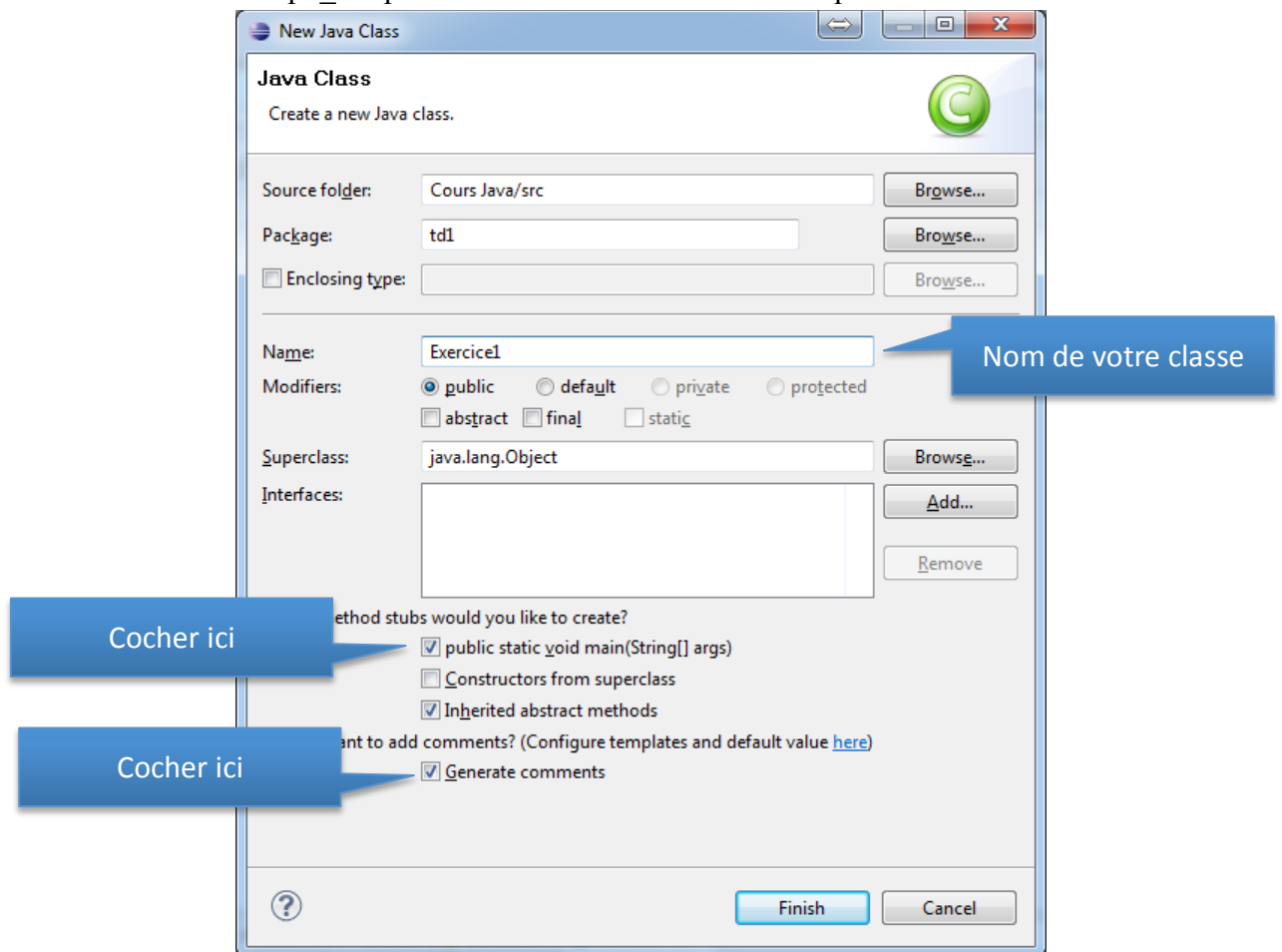


(Pour JRE, Java 1.6, 1.7, ou bien 1.8 sont tout possibles.)

4. Vous devriez présent voir, à gauche dans la colonne « Package Explorer », un dossier « Cours Java » : c'est votre projet. Lorsque vous l'ouvrez, vous voyez qu'il contient un dossier src (qui contiendra les sources de votre programme) et un élément « JRE System Library [J2SE-1.6] » qui liste toutes les dépendances de votre projet lorsque vous l'ouvrez. Nous ne nous préoccupons à présent que du dossier « src ».
5. Faites un clic droit sur « src » et choisissez New -> Package afin de créer un nouveau paquetage. Laissez « Source Folder » tel qu'il est et entrez « tp1_1 » dans Name.

Cliquez ensuite sur Finish. Par convention, les noms de paquetages commencent toujours par une minuscule.

- Un nouvel élément  est apparu dans src : c'est votre nouveau paquetage. Faites un clic droit sur « tp1_1 » puis choisissez New -> Class. Remplissez le menu comme suit :




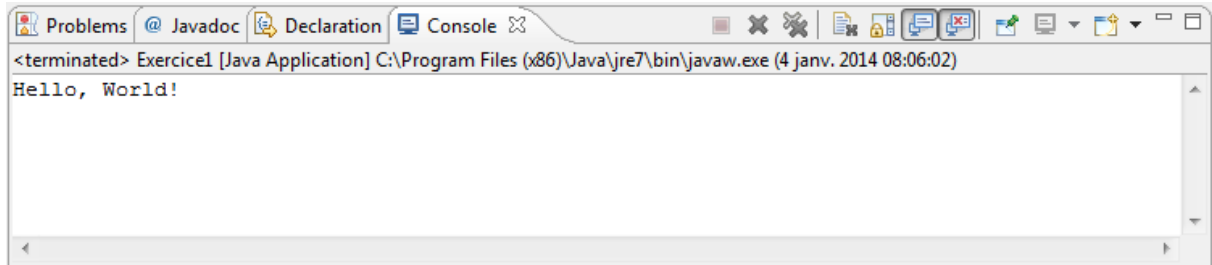
La fenêtre principale vous affiche désormais le contenu de la classe Exercice1. Vous y trouvez également des commentaires de type Javadoc. Tout en haut du fichier, il est inscrit que la classe appartient au paquetage tp1_1. Enfin, une méthode « main » a été créée pour vous. Cette méthode particulière est le point d'entrée de votre programme : c'est à partir d'ici que son exécution débutera.

- Ajoutez à la méthode main le code suivant :

```
System.out.println("Hello, World!");
```

Vous devriez remarquer, lorsque vous aurez tapé "System." Qu'une fenêtre d'autocompétion s'affiche pour vous aider : elle contient tous les champs et méthodes accessibles depuis System. System est une classe appartenant au paquetage « java.lang ». Vous pourriez également écrire « java.lang.System.out... ». System contient différents membres publics, dont « out » pour la sortie standard (par défaut, l'affichage à l'écran), « err » pour la sortie d'erreurs et « in » pour l'entrée standard (par défaut, saisie au clavier). « out » est une instance d'une classe nommée PrintStream. « println » est une méthode de l'objet « out », qui a donc été définie dans la classe PrintStream. Cette méthode affiche un texte et le termine par un retour à la ligne. Comme vous le voyez, on retrouve tous les concepts vus en cours.

8. Cliquez sur le bouton  « Run ». Si le programme vous demande de sauvegarder votre fichier, cliquez sur OK (et pensez à sauvegarder la prochaine fois). En bas de l'interface, devrait s'afficher dans l'onglet « Console » le texte « Hello, World ! ». Si



c'est le cas, félicitations, votre premier programme Java fonctionne correctement !

Exercice 2 : Première classe

Dans cet exercice, nous allons créer notre première classe (Exercice1 ne servant qu'à héberger un point d'entrée au programme), en créer plusieurs instances et afficher leur contenu.

1. Créer une classe *Point*, dans le package `tp1_1`. Ce point doit comporter comme propriétés un nom, sous forme de caractère, une abscisse et une ordonnée, sous forme de doubles. Ces trois propriétés doivent être **privées**.
2. Créer un constructeur prenant les paramètres nécessaires pour initialiser le point. On veut pouvoir écrire :

```
Point pointA = new Point('A', 4.25, 2.75);
```

3. Créer une méthode *affiche* affichant à la console les coordonnées du point. On voudrait comme résultat :

```
Point <nom>, abs=<abscisse>, ord=<ordonnée>
```

La méthode *affiche* doit-elle être publique, protégée ou privée ? Doit-elle être statique ?

4. La méthode *affiche* ressemble beaucoup à une méthode que nous avons discuté en cours : *toString*. Vous pouvez surcharger *toString* en effectuant un clic droit dans le code source de la classe *Point*, puis en choisissant « Source » puis « generate toString() ». Essayez. Quelles sont les options que l'on vous propose ? Quelles différences voyez-vous entre *affiche* et *toString* ?

A quelle méthode de la classe *Objet* cette méthode ressemble ?

5. Créer une méthode *translate*, qui prend en arguments `tx` et `ty`. Cette méthode doit traduire l'abscisse et l'ordonnée de l'instance d'un point de `tx` et `ty` respectivement.
6. Comment traduire un point A de façon à ce que ses coordonnées soient les mêmes que le point B ? Ajouter des méthodes *getAbscisse* et *getOrdonnee* pour aider à y remédier.
7. Créer une méthode *distance*, qui prend en argument un point `p2` et renvoie la distance entre l'instance courante et le point `p2`. Pour information la racine carrée s'écrit `Math.sqrt()`. Vous remarquerez que les propriétés d'abscisse et d'ordonnée de `p2` sont accessibles à l'intérieur de la méthode *distance*.

8. Créer une méthode *confondu*, qui renvoie vrai si et seulement si le point courant et le point passé en paramètre sont situés aux mêmes coordonnées. Vous utiliserez pour cela la fonction *distance* plutôt que comparer les coordonnées.
9. Dans la fonction *main()* (celle de l'exercice 1 par exemple, ou copiez-la dans la classe *Point*), instanciez les points suivants :
A(4, 2.75)
B(0, 0)
C(-5, 2.75)
D(0.0001, 0.0001)
10. Quelle est la distance entre les points A et C ? Les points B et D sont-ils confondus ? Utilisez les fonctions créées précédemment pour le vérifier.

Exercice 3 : Aller plus loin avec les nombres (facultatif)

Dans cet exercice, nous allons améliorer notre classe *Point* pour prendre en compte des cas potentiels d'erreur.

1. Certaines valeurs spécifiques sont représentables avec des doubles : les infinis positifs et négatifs ainsi qu'une valeur spéciale, NaN (not a number – pas un nombre). Ajouter à la fonction *translate* une vérification que les doubles passés en argument ne correspondent pas à l'un de ces trois cas (*Double.POSITIVE_INFINITY*, *Double.NEGATIVE_INFINITY*, *Double.NaN*). La méthode ne doit pas modifier l'abscisse et l'ordonnée si un de ses deux paramètres est invalide et retourner la valeur booléenne *false*. En cas de succès, elle retournera vrai. Vous remarquerez qu'il n'est pas possible de faire de même avec *distance*, qui renvoie déjà un double, ou avec le constructeur, qui n'a pas de valeur de retour.
2. De par leur représentation interne, certaines valeurs ne sont pas représentables exactement avec double et float. Vous pourrez le vérifier simplement en faisant une boucle *for* ajoutant 10 fois à un compteur la même valeur flottante 0.01f. Le résultat devrait être 0.099999 et non 0.1.
Pour pallier ce problème, créez une valeur double appelée *EPSILON* qui soit commune à toutes les instances de *Point*. On lui donnera par exemple comme valeur 0.0001. *EPSILON* doit-il être publique, privé, statique, constant (final en Java) ?
Modifiez la méthode *confondu* pour vérifier, non plus que la distance entre deux points est zéro, mais que celle-ci est inférieure à *EPSILON*. Les points B et D sont-ils maintenant égaux ?

Exercice 4 : Composition de classes

Dans cet exercice, nous allons voir comment une classe peut en réutiliser d'autres en tant que membres, et utiliser leurs fonctions.

1. Créez une classe Triangle qui possède comme membres trois points pointA, pointB et pointC de type Point. Vous prévoirez un constructeur qui prenne en paramètre trois points pour les assigner comme valeurs aux membres.
2. Ecrivez une méthode *confondu* qui prenne en paramètre un second triangle et vérifie si les deux triangles sont confondus (pensez à toutes les variantes possibles).
3. Est-ce que on peut enlever Point.java après la compilation de Point.java et créer la class Triangle en utilisant la Class Point comme ci-dessus ?

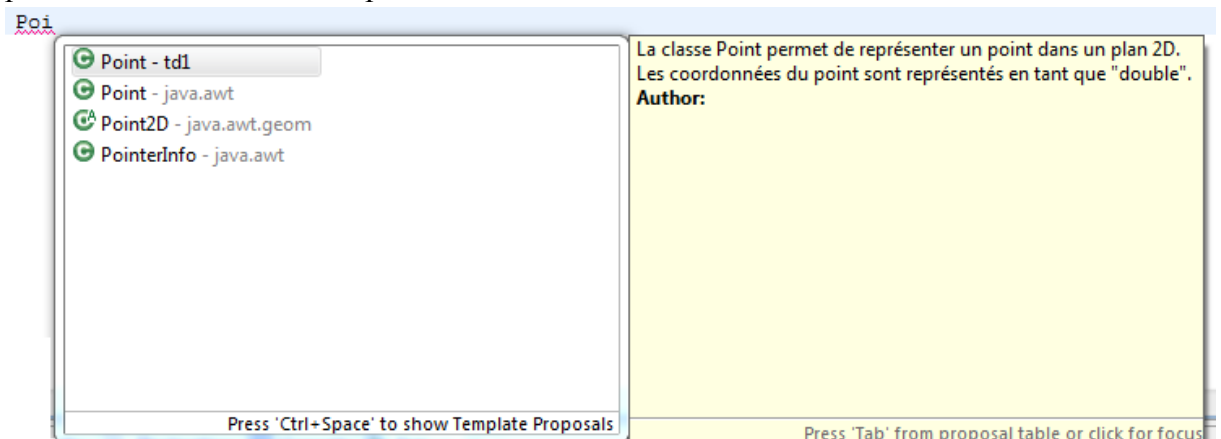
Exercice 5 : Construire la documentation

Dans cet exercice, nous allons voir comment construire la documentation Javadoc pour aider à la réutilisation de notre classe.

1. Sur la ligne précédant « public class Point », tapez la commande suivante : « /** », puis appuyez sur la touche retour à la ligne. Un nouveau commentaire multiligne a été créé.
2. Remplissez le commentaire avec les informations suivantes :

```
/**
 * La classe Point permet de représenter un point
 * dans un plan 2D. Les coordonnées du point sont représentés
 * en tant que "double".
 * @author <votre nom>
 *
 */
```

3. Créez une nouvelle classe nommée « Exercice5 », possédant un point d'entrée au programme (méthode main publique et statique). Dans la méthode main, tapez « Poi » puis les touches contrôle+espace. Les informations suivantes devraient s'afficher :



Vous remarquerez qu'il existe également une classe Point dans java.awt.

Terminez la ligne en écrivant par exemple « Point p = null ; », l'objectif est simplement de vous faire expérimenter la Javadoc.

4. Effectuez les mêmes manipulations qu'en question 1 sur toutes les méthodes de la classe Point. Vous remarquerez que certaines lignes apparaissent automatiquement, comme « @param » et « @return ». A quoi servent-elles ? Quelles sont les autres options existant ? (commencez une ligne par un @ pour le découvrir)

Exercice 6 : Des types plus précis (facultatif)

Les types float et double ont des précisions limitées, du à leur besoin de rapidité de calcul. Dans cet exercice, vous allez voir comment utiliser un type dont la précision peut être garantie : BigDecimal.

1. Reprenez les exercices 2 à 4 en remplaçant les types « double » par des types « BigDecimal ».

Pour instancier un BigDecimal, il vous est conseillé d'utiliser la syntaxe suivante :

```
new BigDecimal("4.25");
```

Les opérations de base s'effectuent en appelant les méthodes add(), subtract() etc.

Pour choisir la précision (nombre de chiffres après la virgule), vous pouvez utiliser la méthode scale(). Cette méthode vous sera parfois nécessaire car la précision varie selon les méthodes utilisées : multiply(), par exemple a comme précision la somme des précisions des deux opérandes.

Vous trouverez plus d'informations dans la documentation Java :

<http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html>

Exercice 7 : Construire la documentation

Fermez Eclipse. En suite, compilez et exécutez les classes que vous avez créées par la ligne de commande.