Semaine méthodes

Matière

- Les méthodes

Objectifs

- Comprendre les différents objectifs visés en introduisant des méthodes
- Comprendre le principe du passage de paramètres et de l'appel de méthode
- Pouvoir écrire un if imbriqué dans une méthode
- Savoir procéder à une découpe en méthodes : une boucle par méthode !
- Comprendre le rôle de la *JavaDoc*

Pour les exercices de cette semaine, créez un nouveau projet IntelliJ et appelez-le ALGO_methodes.

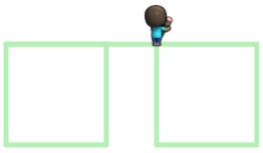
Exercices obligatoires

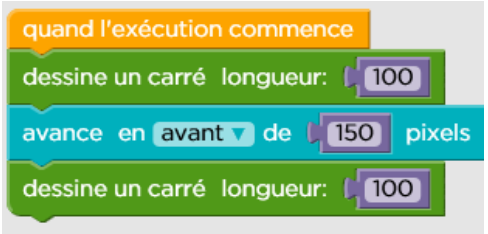
A La tortue : le retour !

DRY (Don't repeat yourself) - Ne vous répétez pas !

La factorisation de code consiste à rassembler les suites d'instructions identiques en une « méthode » pour améliorer la lisibilité et en faciliter la correction et les modifications ultérieures.

Ce sont toujours les mêmes formes géométriques qui apparaissent dans les dessins que vous avez programmés avec la tortue : des carrés, des triangles, des cercles, …
Voilà une belle impression de « copier-coller » de mêmes suites d'instructions au sein d'une même classe et aussi d'une classe à l'autre !





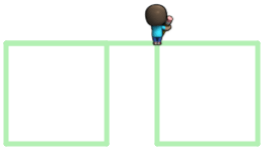
Donnons à la tortue plus de responsabilités :

```
tortue.
    dessinerUnCarre(double longueur)
    dessinerUnTriangle(double longueur)
    accelerer()
    avancer(double pixel)
    tournerADroite(int degre)
    definirCouleur(String couleur)
    tournerAGauche(int degre)
```

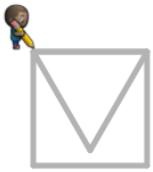
A1 La classe *Tortue* fournie cette semaine contient déjà la méthode `dessinerUnCarre(double longueur)`.
Ajoutez-lui la méthode `dessinerUnTriangle(double longueur)`.

En utilisant les méthodes `dessinerUnCarre()` et `dessinerUnTriangle()` qui ont été ajoutées dans la classe *Tortue* :

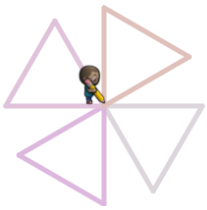
A2.1 Ecrivez la classe *DessinLunettes*



A2.2 Ecrivez la classe *DessinEnveloppe qui* dessine une enveloppe.



A2.3 Ecrivez la classe *DessinFleur*



A2.4 Ecrivez la classe *DessinVentilateur*.

A2.5 Ecrivez la classe *DessinCarresEmboités*.

B La classe *Utilitaires*

Technique Bottom-up : on commence par réaliser toute une série d'outils qui pourront servir pour élaborer différents programmes.

Nous ne comptons plus le nombre de fois qu'on a eu besoin de copier-coller la méthode `unEntierAuHasardEntre()`.

Jusqu'à présent, nous acceptons les entrées clavier sans jamais les vérifier. Que se passe-t-il si on encode un prix négatif ? Une cote supérieure à 20 ? Un diviseur égal à 0 ?

Nous avons décidé de rassembler dans une même classe des **méthodes utiles à de nombreuses classes**. Nous avons appelé cette classe ***Utilitaires***.
Au fur et à mesure de l'année, il serait intéressant de l'avoir sous la main et de penser à la compléter.

Voici quelques exemples d'utilisation de méthodes déjà fournies :

```
int lanceDe = Utilitaires.unEntierAuHasardEntre(1,6);

double taille = Utilitaires.lireReelPositif();
```

B1 Complétez les méthodes `lireReelComprisEntre()` et `lireOouN()` de la classe *Utilitaires*. Améliorez la classe *CalculMoyenne* en validant toutes les lectures clavier.

B2 Revoyez la classe *DivisionEntiere*. Il faut éviter les divisions par 0 !
Commencez par ajouter la méthode `lireEntierNonNul()` dans la classe Utilitaires et utilisez-là.
N'oubliez pas d'ajouter la *JavaDoc*.

B3 Revoyez la classe *Statistiques*. On voudrait que toutes les lectures clavier soient validées.
Commencez par compléter la classe *Utilitaires*.

C Les ifs imbriqués

Le *return* provoque la sortie de la méthode.
Il ne faut plus faire apparaître les *else* dans les *if*s imbriqués.

Observez la méthode `donnerEtat()` de la classe *CalculBMIV2*.
Cette classe est une version avec méthodes de la classe *CalculBMI* de la *ficheIf*

C1 Ecrivez une nouvelle version de la classe *RechercheMax3* en introduisant une méthode `max3()`. A vous d'écrire l'entête de la méthode. Réfléchissez bien aux paramètres et au type de retour. Appelez cette version, *RechercheMax3V2*.

C2 Ecrivez une nouvelle version de la classe *CoteCommentee* en introduisant une méthode `donnerCommentaire()`. A vous d'écrire l'entête de la méthode. Réfléchissez bien aux paramètres et au type de retour. Appelez cette version, *CoteCommenteeV2*.

D Les boucles imbriquées

Les instructions que l'on place à l'intérieur d'une répétitive peuvent être de tout type, en particulier il peut s'agir d'autres répétitives. On obtient ainsi des boucles imbriquées dans d'autres boucles.
Souvent, l'usage d'une méthode permet de cacher une telle imbrication.

Technique Top-down : on pense d'abord au programme principal en laissant pour plus tard l'étude des détails, que l'on confiera à des méthodes

D1 Concevez un programme qui permet de donner quelques informations sur la classe délibérée.
La classe compte 25 étudiants.
Pour chaque étudiant, le programme lit au clavier ses différentes cotes (sur 20), calcule et affiche sa moyenne.
Le nombre de cotes est 10. Il est le même pour chaque étudiant.
A la fin, le programme affiche la moyenne de la classe.

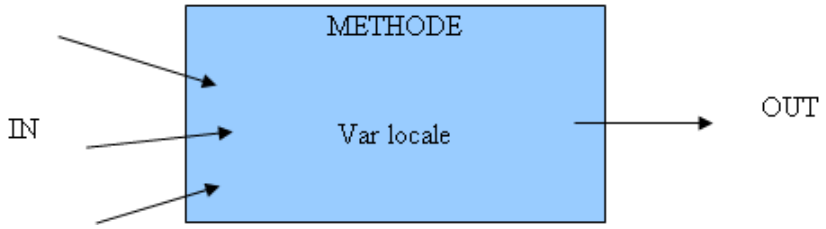
Vous n'allez pas directement programmer !

Dans un premier temps, on réfléchit à une découpe en sous-méthodes !
On exprime, en français structuré, les principales étapes de l'algorithme (macro-code) :

« On boucle sur les étudiants » et pour chaque étudiant, « on boucle sur ses cotes ».

Les boucles sont imbriquées. Pour cacher une telle imbrication, on va introduire une méthode. Cette méthode contient la « boucle sur les cotes » et le `main()` contient la « boucle sur les étudiants ».

Ecrivez l'entête de cette méthode.
Il faut lui trouver un nom.
Il faut se pencher sur les échanges d'informations entre le `main()` et cette méthode.



IN : La méthode a-t-elle besoin d'informations provenant du `main()` ?
OUT : Quelle information le `main()` attend-elle de cette méthode ?

Ecrivez la classe *Deliberation.java*.
Pensez à utiliser la classe Utilitaires.
Donnez la *JavaDoc* de la méthode.

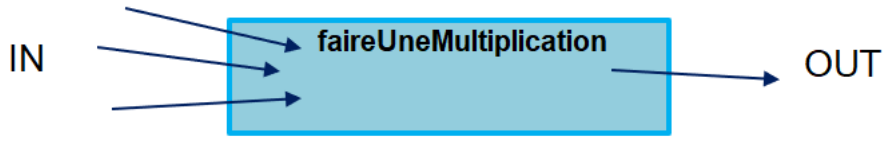
E Les classes *Multiplication1*, *Multiplication2*, …

Les 4 programmes « *Multiplication* » sont différents mais possèdent la même caractéristique : proposer **plusieurs** exercices de multiplication !

On « imagine » facilement pour chacun de ces programmes introduire une méthode qui s'appelle `faire`**`Une`**`Multiplication()`. Cette méthode, comme son nom l'indique, s'occupe d'une multiplication. La méthode `main()` contiendra la répétitive dans laquelle se fera l'appel à cette méthode.

On ne vous demande pas d'écrire tous ces programmes !
Dans chacune des classes, on vous demande d'ajouter l'entête de la méthode `faireUneMultiplication()` **en Java** (*JavaDoc* comprise).

Commencez par réfléchir aux échanges d'information entre la méthode et le main() en remplissant le schéma suivant :



Exercices supplémentaires

D2 Aux éliminatoires du championnat du monde de gymnastique, chaque concurrent est coté pour la présentation de son exercice, sur 10 points, par 8 membres du jury.
On élimine de ces cotes la plus haute et la plus basse. Le résultat d'un concurrent est égal à la moyenne des 6 cotes restantes.
Pour passer en finale, il faut un résultat d'au moins 8/10.
Écrivez un programme qui passe en revue les différents concurrents.
Pour chaque concurrent, le programme lit les notes du jury et affiche son résultat.
Après chaque concurrent, le programme demande s'il y en a un autre, via le message :
« Encore un concurrent (O/N) ? »
A la fin, le programme affiche le nombre de participants qui vont en finale.

Dans un premier temps, on vous demande, d'écrire un macro-code.
Vous décrirez en français structuré les principales étapes de l'algorithme.
Procédez à une découpe en méthodes.

Ensuite prenez la classe *Championnat.java* et complétez-la.
Pensez à utiliser la classe Utilitaires.

La méthode `resultatUnParticipant()` lit les cotes (entiers) d'un concurrent, calcule et renvoie son résultat (réel).
Vous avez peut-être choisi la méthode `estQualifie()`, c'est également correct.
Cette méthode qui renvoie un booléen suffit pour répondre à la question : combien y a-t-il de finalistes ? Mais elle ne permet que cela !
La méthode `resultatUnParticipant()` permet de répondre à cette question mais permet plus comme par exemple, afficher la moyenne des concurrents, le meilleur résultat, ...

E Complétez les classes « Multiplication ».

Exercices défis

C4 Dans la classe *Utilitaires*, ajoutez une méthode `public static char lireCharPermis(String caracteresPermis)` qui demande à l'utilisateur d'entrer un caractère au clavier. Ce caractère devra être contenu dans la chaîne de caractère passée en paramètre à la méthode. Si l'utilisateur se trompe, on lui repose la question jusqu'à avoir une bonne réponse. La méthode renverra le caractère entré par l'utilisateur.

Utilisez cette méthode dans la classe *NombreMystereInverse* écrite en la semaine *while*.