Bonjour, je vais vous présenter dans cette vidéo comment installer les logiciels qui vous permettront très facilement d’écrire, compiler et exécuter des programmes java. Cette solution c’est si vous pouvez installer des logiciels sur votre ordinateur. Si vous ne pouvez pas, ou si le dispositif que vous avez est plutôt une tablette, il y a la solution doppio que je présente dans une autre vidéo et qui vous permet de travailler sans installer de logiciel mais avec des conditions de confort moindres. Donc je vous propose de commencer par créer un nouveau dossier pour y mettre tous vos programmes du MOOC. Donc on peut l’appeler "MOOC java" si vous avez un autre nom que vous préférez, vous pouvez prendre celui que vous voulez. Et là on va avoir trois outils à télécharger : le compilateur java, l’environnement de développement et la librairie MOOC. Le compilateur on va aller le chercher chez Oracle qui est le propriétaire du langage java et qui donc fait un compilateur gratuit qui est celui de référence que je vous conseille ; l’environnement de développement c’est DRjava ; et la librairie MOOC c’est quelque chose de spécifique à ce MOOC qu’on ira chercher sur France université numérique. Pour trouver le compilateur, je vous propose d’aller sur Google et de taper les mots clés jdk download, et le premier lien est le bon, vous vérifiez que c’est bien chez oracle.com et vous cliquez donc sur le premier lien qui s’affiche. Il vous propose java se download, vous cliquez sur l’icône java ici, et la première chose à faire c’est d’accepter la licence. Une fois la licence acceptée, vous pouvez choisir la version qui convient à votre machine, vous avez ici la version mac, les versions Windows et les versions pour linux ou Solaris. Ma machine est une machine Windows 64, donc je vais télécharger le jdk pour le Windows 64, il y a une petite minute de téléchargement. Donc pendant que cela se fait, je vais aller chercher l’environnement de programmation DRjava. Donc je tape en un seul mot DRjava et là à nouveau, le premier lien est le bon, c’est l’url www.drjava.org, il me propose la version stable courante "current stable release", c’est ce qu’il me faut et il me propose trois versions : une version mac, une version Windows et une version pour les autres systèmes. Je choisis la version qui correspond à ma machine, c’est du Windows. Là il y a un petit délai et au bout d’une seconde le téléchargement commence. Celui-là est rapide, il dure quelques secondes, c’est un logiciel beaucoup plus léger quoique là il a l’air de se stopper, on va le laisser finir tranquillement. La librairie MOOC c’est une librairie spécifique que vous allez trouver sous la vidéo dans les supports de cours ici. Il y a la vidéo et il y a le fichier mooc.java. Je clique dessus, il s’affiche dans le navigateur. Si je fais un clique droit, je peux faire enregistrer sous et je vais l’enregistrer dans le dossier que j’ai créé donc je vais aller chercher sur le bureau le dossier que j’ai appelé mooc.java. J’enregistre, voilà il doit être au bon endroit. Pendant ce temps, les téléchargements se sont faits et se sont bien terminés. Je peux aller dans mon navigateur sur les pages de téléchargements ou alors dans l’explorateur Windows. J’ai deux choses différentes à faire ici, il faut que j’installe le compilateur donc le jdkmachintruc.exe, je clique dessus pour l’exécuter. Là mon Windows me dit que ça peut être une opération risquée d’installer mais je vais le faire quand-même. Et là on rentre dans la procédure d’installation, et ce que je vous conseille c’est surtout de ne pas contrarier java, c’est-à-dire de faire toujours ce qu’il a envie de faire, en clair d’appuyer toujours sur le bouton next, c’est ce que je veux faire maintenant. Voilà, donc là il va passer un certain temps à copier des fichiers et à installer tout ce qu’il faut, c’est un gros logiciel donc ça prend un certain temps. Pendant que… là une petite pub qui nous dit qu’il y a trois milliards de dispositifs dans le monde qui utilisent java, donc bientôt un de plus avec cet ordinateur, trois milliards et un, enfin je ne pense pas que le degré de précision soit celui-là, c’est à la louche trois milliards de dispositifs, ils ne disent pas d’ailleurs combien dans les trois milliards sont dans les décharges et les poubelles. Le téléchargement et l’installation sont terminés, c’est bon. La dernière chose que je vais faire c’est de transférer le fichier que j’ai téléchargé DRjava stable .exe, je vais le mettre sur le bureau pour pouvoir l’utiliser facilement. Voilà, il est sur le bureau, je peux fermer mon navigateur, - je ne sais pas ce qu’il est entrain de télécharger -. Là, normalement tout est terminé, je suis en position de travailler, je vais pouvoir utiliser les outils mais ça c’est le sujet d’une autre vidéo.

Bonjour nous allons voir dans cette vidéo comment utiliser DRjava pour écrire, compiler et exécuter les programmes, vous allez voir c’est très simple. Je suis dans un état où j’ai laissé l’ordinateur à la fin de l’installation donc il faut avoir au préalable réalisé l’installation qui est décrite dans une autre vidéo.

Je commence en cliquant sur DRjava que j’avais mis sur le bureau si vous vous rappelez, il faut un petit peu de temps pour qu’il s’ouvre. DR java, on voit trois zones : une zone là où vient agir le curseur qui clignote et où je vais commencer à écrire mon programme, une zone ici où je vais avoir les noms des fichiers que j’ai ouverts, et ce qui me permettra de naviguer d’un fichier à l’autre et ici c’est en bas une zone où je vais avoir des informations diverses et c’est là aussi que se fera l’exécution des programmes.

Je vais commencer à taper le programme "patate", je ne vais pas le taper entièrement, je vais taper le début "public static void main", la petite incantation qu’il faut connaître sans la comprendre et après les instructions. Je déclare le nom poids, je fais des affichages system.out.println. Je vais chercher au clavier un certain nombre, double, et puis qu’est-ce que je peux faire ? Je peux l’afficher voilà. Donc je vais me contenter de ça, je ne tape pas tout le programme pour aller plus vite. J’ai fini de taper le programme, il faut que je le sauve dans un fichier, pour cela j’ai un bouton ici. Mais en fait toutes les opérations importantes sont dans les boutons que j’ai ici en haut et donc pour sauver c’est « save ». Il me propose un nom de fichier qui est le bon, mon programme s’appelle" exemple", j’ai juste à faire enregistrer. Alors il n’est pas au bon endroit, parce que j’ai oublié de le sauver dans le dossier mooc java voilà. Cette fois je l’ai mis dans le bon dossier mon programme. Et voyez que le nom du programme apparaît à gauche dans la fenêtre des noms de fichiers. Le programme est tapé, il faut maintenant que je le compile, la compilation c’est la même chose, j’ai un bouton ici qui permet de faire la compilation et quand tout se passe bien, le message qui s’affiche c’est "compilation completed". Si je fais une erreur par exemple j’oublie un point virgule, je ressauve, je compile, ce que j’ai c’est un message d’erreur, une "erreur found" et j’ai en jaune le message d’erreur et la ligne sur laquelle il pense qu’il y a une erreur. Alors, il faut savoir que le compilateur parfois se trompe, il vous dit que l’erreur est là et ça peut être en fait à la ligne précédente ou ailleurs. Mais souvent il ne se trompe pas ici il a bien raison, il bien a vu qu’il manquait un point-virgule, il me dit point-virgule explicated et il me donne la bonne ligne donc je rajoute mon point-virgule. Avant de le rajouter, je vais vous montrer ce qui se passe quand il y a deux erreurs, je sauve, je compile, il y a deux erreurs, il m’en met une en jaune et la ligne correspondante en jaune. Si je veux voir l’autre erreur, je peux cliquer dessus et c’est l’autre qui passe en jaune.

Je rajoute mes points-virgules parce que je ne veux pas d’erreurs, alors des erreurs ce n’est pas grave d’en faire, quand on tape des programmes cela arrive de faire des erreurs, des fautes de frappe d’oublier quelque chose etc. donc les erreurs on en fait ce n’est pas grave, ce qui est important c’est de savoir les corriger.

Je compile, j’ai corrigé mes erreurs, la compilation s’est bien passée. Pour exécuter le programme, j’ai le bouton "run", c’est parti, il commence à exécuter dans la fenêtre du bas. Il me demande de rentrer donc un poids, je le tape ici "entrée" et il l’affiche donc vous voyez l’exécution s’est bien passée, elle s’est terminée, je peux refaire une autre exécution si je le souhaite et ça marche. Donc on a vu les boutons les plus importants qui sont : "save ", "compile" et "run" les trois boutons essentiels.

" New" c’est si je veux faire un nouveau programme. Je fais un deuxième programme rapidement, petit programme juste pour avoir un deuxième fichier et voir ce qui se passe, je le sauve, il est bien dans le dossier "mooc java", il a bien le bon nom. Et on voit que donc j’ai mes deux noms de fichiers qui s’affichent et que je peux passer de l’un à l’autre en cliquant. Je clique sur exemple, j’ai le programme exemple, je clique sur truc, j’ai le programme truc. Je l’ai sauvé, je peux le compiler, l’exécuter.

"Open" c’est si je veux ouvrir un fichier que j’ai déjà fait une fois précédente où j’ai utilisé DRjava. Je peux par exemple ouvrir mooc.java.

"close" c’est si je veux fermer un fichier, il disparaît de la liste. Je ferme truc, il me reste plus que exemple, je ferme exemple, il ne me reste plus rien.

"Cut-copy-paste" c’est le copier-coller que vous connaissez, copier-coller habituel mais pouvez aussi bien utiliser vos raccourcis clavier, Ctrl-C, Ctrl-V ou pomme C, pomme V. Find, c’est pour trouver quelque chose, compile, run, on a fait le tour des fonctions essentielles. Après il y a les menus, si par exemple on veut changer le nom on fait un save-A. Globalement vous aurez très rarement à aller voir les menus, et il suffira d’utiliser les boutons qui sont là.

Voilà, donc vous voyez, c’est un outil très simple à utiliser qui permet de tout faire : écrire les programmes dans la fenêtre ici, les compiler bouton" compile", les exécuter bouton "run". Donc maintenant vous avez les cartes en mains pour jouer, pour faire les programmes et notamment le premier test hebdomadaire qui vous attend tout de suite. Bon travail.

Djava

Bonjour, je vais vous présenter l’environnement de programmation doppio qui vous permet d’écrire, compiler et exécuter des programmes java dans votre navigateur et c’est la solution à privilégier si vous ne pouvez pas installer des logiciels comme DRjava ou des compilateurs sur votre ordinateur.

Le confort est moindre mais cela vous permet de travailler. Donc voici la fenêtre avec le navigateur qui peut être Chrome, Firefox, ou Internet Explorer ou Safari, et si je veux créer un nouveau programme je vais appuyer sur le bouton nouveau et il me demande le nom du programme. Je vais l’appeler nouveau, " nouveau.java", et il me crée les deux petites lignes qu’on ne comprend pas bien là avec le nom du programme et public static void main, et là je peux taper mon programme. Donc si je tape le début du programme patate, je peux déclarer le nom poids system.out.println, quel poids, - j’ai oublié un i ici- poids = mooc.readDouble et puis je peux afficher system.out.println fini. Ce n’est pas tout à fait le programme patate, c’est juste pour redémontrer ; donc une fois que j’ai tapé mon programme, je l’enregistre et je lance la compilation. La compilation est un peu lente, on voit ici le petit symbole qui nous dit que c’est entrain de travailler. Donc c’est à droite de l’écran, ça tourne et on espère que la compilation va bien se terminer ; ça peut mal se terminer, ça voudrait dire qu’il y a un message d’erreur. Voilà, donc il y a un message d’erreur parce qu’au lieu d’écrire readDouble, j’ai écrit readdoubel. Donc je vais modifier, corriger mon programme, pour cela vous voyez qu’il y a son nom qui apparaît ici, je vais demander à l’ouvrir dans l’éditeur, et donc je vais changer doubel par double, je ressauve et je recompile. Alors c’est un petit peu lent parce que ça s’exécute dans le navigateur, mais là ça s’est bien terminé, la compilation est terminée. Je vais maintenant demander l’exécution et je change un peu le cadrage, puisqu’il va falloir que je rentre les choses un peu en dessous, exécuter, ça affiche donc les affichages en haut mais les entrées il faut que je les mette ici, donc 3.15, voilà, le programme est terminé et il nous dit l’exécution s’est terminée avec succès.

Donc on a vu comment faire un nouveau programme, l’enregistrer, le compiler, l’exécuter, maintenant il est possible notre programme de le sauvegarder sur l’ordinateur, là il est dans le navigateur, on peut le télécharger sous-forme d’un fichier, j’appuie ici et le programme "nouveau.java" est arrivé sur le disque de l’ordinateur. Je peux dans l’autre sens aller chercher sur l’ordinateur un fichier pour le mettre sur doppio, je vais dans terjava, je vais prendre par exemple patate voilà, le voilà et voilà mon programme patate dans doppio, et je peux le compiler et l’exécuter. La compilation est terminée, l’exécution est partie 4.3.5, voilà, l’exécution s’est terminée avec succès. Voilà donc, Une petite mise en garde c’est que là donc j’ai mes deux programmes, je peux aller de l’un à l’autre, ouvrir nouveau, ouvrir patate. Malheureusement si je recharge la page là, si je fais actualiser la page, je perds tous mes programmes, je le fais je n’ai plus rien, il a tout oublié. D’où l’intérêt de pouvoir sauvegarder sous-forme d’un fichier et enregistrer et de pouvoir utiliser de doppio vers l’ordi et de l’ordi vers doppio. Donc avec ça vous allez pouvoir faire tous les programmes qu’on vous demandera de faire dans ce MOOC.

Invite de la semaine

Bonjour, on voit à l’écran une photo de la place de la concorde à Paris, avec l’obélisque au centre, mais l’obélisque qu’on voit de loin donc on ne peut pas bien lire ce qu’il y a dessus, on va passer sur une photo en gros plan qui montre un petit bout de l’obélisque. Mais moi je ne sais pas lire ces choses-là, et donc j’ai demandé de l’aide.

Serge ROSMORDUC : Bonjour, je suis Serge ROSMORDUC enseignant d’informatique au CNAM, mais ma spécialité ce sont les humanités numériques et en particulier le traitement du langage pour l’égyptologie. Donc je sais lire les hiéroglyphes en particulier, et donc là vous avez un bout de l’obélisque de la concorde et l’obélisque de la concorde c’est assez simple : dessus il y est marqué Ramsès II. Ramsès II était un souverain connu pour sa modestie comme la plupart des pharaons Egyptiens d’ailleurs et il écrivait son nom très profond partout pour que ses successeurs ne puissent pas l’effacer. Donc là vous avez sur l’obélisque de la concorde un petit passage où il est marqué en gros Ramsès. On voit qu’il ya le même texte à gauche et à droite, donc qui écrit le fils de Rê, là le petit canard avec le rond, le canard c’est fils et puis le rond c’est Rê, donc le fils de Rê, Rê c’est le dieu soleil et en dessous vous avez écrit Ramsès aimé d’Amon. Et au centre vous avez un autre texte un peu différent avec toujours le canard qui écrit toujours fils, et ça écrit le fils ainé du roi des dieux, le roi des dieux c’est Amon qui est aussi le dieu principal au même titre que Rê, c’est un peu le même et donc sur l’obélisque vous avez écrit Ramsès, Ramsès, Ramsès. Et puis vous avez des épithètes qu’il protège l’Egypte, qu’il casse la figure des voisins et qu’il fait des temples pour faire plaisir à son papa le dieu Amon, voilà.

François BARTHELEMY : bien, je t’ai demandé de venir nous parler des hiéroglyphes, parce que tu as écrit un logiciel en java pour taper les hiéroglyphes donc ce qu’on peut appeler un traitement de texte pour les hiéroglyphes que tu vas nous montrer.

Serge ROSMORDUC : oui donc ce logiciel c’est "JSesh" qui est un traitement de texte qui est utilisé par les égyptologues quand ils veulent mettre des hiéroglyphes dans leurs articles et dans leurs publications. Je vais vous montrer un peu comment je vais taper un petit bout du texte qu’on a vu précédemment, donc je vais taper simplement le fils de Rê Ramsès. Donc pour taper, je vais d’abord dire que mon texte va être un texte en colonne et de droite à gauche, c’est la disposition qu’on a dans notre exemple et puis je vais saisir maintenant le texte. Donc le texte, je le saisis de façon à moitié phonétique, donc par exemple fils ça se dit ça et donc je vais taper, ça ne se voit pas bien mais j’ai tapé ça. Je tape sur la barre d’espace et j’ai un signe qui n’est pas le bon, alors comme parce qu’ils y a plusieurs signes qui peuvent se prononcer ça, et donc je vais taper à nouveau sur la barre d’espace pour circuler entre les signes, c’est une façon de faire qui est utilisée par exemple pour le chinois ou le japonais aussi. Maintenant je tape "ra" et j’avais demandé de ligaturer les dossiers pour qu’ils se placent bien, et puis je vais commencer à écrire Ramsès, donc pour cela j’ai besoin de choisir les hiéroglyphes des dieux, donc d’abord j’ai le petit "si" avec le dieu Ra à si qui était inversé, alors ça c’est simplement graphique c’est pour qu’il soit en face du dieu Amon qui lui fasse la causette, ensuite j’avais besoin du dieu Amon, là ils sont l’un en dessous de l’autre je vais les mettre côte à côte. Et puis ensuite je vais écrire aimer, et la fin de mon texte en les mettant côte à côte à nouveau et je termine en les mettant dans ce qu’on appelle un cartouche qui ensert normalement le nom du roi. Voilà, là j’ai mon Ramsès aimé d’Amon qui a été écrit. Je vais maintenant vous montrer à quoi ressemble l’intégralité de l’inscription si je la charge dans JSesh, là vous voyez que j’ai le texte qu’on voyait précédemment qui apparaît.

François BARTHELEMY : merci donc je crois que ce qu’il y a de bien avec ce logiciel c’est qu’on peut faire du copier-coller facilement donc par exemple dans du Word ou dans un document normal tu peux nous l’illustrer.

Serge ROSMORDUC : oui, je copie donc édition-copier, -ce n’est pas grave- ça a été copié et je vais dans open office et je fais coller et voilà j’ai le texte. Je peux l’agrandir si ça ne me convient pas, et voit que j’ai un très beau dessin qui a été inséré dans mon texte Word.

François BARTHELEMY : et là pour montrer que tu n’es pas le seul à utiliser ton logiciel, nous avons un extrait d’un article d’égyptologie qui intègre donc du français et des hiéroglyphes, et les hiéroglyphes ont été faits avec du JSesh et copiés-collés probablement dans Word enfin, je suppose.

Serge ROSMORDUC : oui

François BARTHELEMY : donc ce logiciel qui est un gros logiciel quand-même, je crois plus de 1000 fichiers différents.

Serge ROSMORDUC : oui et 170 000 lignes de code.

François BARTHELEMY : donc c’est du java, et pour le montrer on a décidé de vous montrer la méthode "Main", parce que vous la connaissez et c’est donc une méthode main comme les programmes que vous écrivez vous. Et on voit que la méthode main est petite mais c’est le reste du programme qui est gros. Donc tu as commencé à faire ce programme il y a quelques années maintenant, et tu as choisi de le faire en java, pourquoi ?

Serge ROSMORDUC : alors, j’ai choisi de faire en java parce que java est portable et que les utilisateurs potentiels de mon logiciel se partageaient à peu près également entre les utilisateurs de mac et les utilisateurs de PC. Donc j’avais besoin d’un langage qui me permette de créer des applications graphiques relativement rapides et qui fonctionnent sur tous les types de machines et qui soient aussi facile à maintenir c’est-à-dire à installer sur machine ou sur une autre. Donc j’ai besoin par exemple de travailler sur PC séparément de sur mac etc. et java permettait cela. Je peux en gros quand je développe, j’écris sur Macintosh et je le copie sur le PC après et il fonctionne. Je n’ai pas besoin de retravailler sur PC pour que ça marche.

François BARTHELEMY : D’accord, donc là c’est un choix que tu as fait il y a une dizaine d’années au début du projet, est-ce que de nos jours il y aurait une alternative à java ? est-ce que tu choisirais toujours java ?

Serge ROSMORDUC : alors il y a pas mal d’alternatives, mais je pense que java de ce côté-là a encore une certaine facilité de développement qui ferait que je resterais sans doute sur java. Il y a d’autres langages qui sont développés, il y a Mono et Csharp plutôt sur le monde Microsoft mais qui peuvent fonctionner sur Mac et vous avez aujourd’hui le langage java script qui est entrain de devenir une sorte de langage général qui va sans doute prendre de plus en plus d’importance.

François BARTHELEMY : d’accord, écoute, c’était intéressant pour nous d’avoir un développeur d’une vraie application java, une application assez largement diffusée, des milliers de téléchargements, des milliers d’utilisateurs, donc c’était intéressant pour nous d’avoir ton témoignage.

Serge ROSMORDUC : si vous voulez jouer avec, elle est gratuite, vous tapez JSesh dans Google et vous pouvez l’utiliser.

Semaine 2

Bonjour.

Nous allons attaquer la deuxième semaine de notre MOOC avec une semaine qui a pour titre Avec des si.

Avant de voir le programme de la semaine, on va déjà faire un petit point rapide sur ce qu’on a déjà vu. On a déjà vu la structure d’un programme avec classe, le nom du programme public static void main, une suite d’instructions. On a vu, parmi les instructions qu’on a déjà vues, comment définir un nom pour une valeur inconnue, avec double poids, comment afficher des messages à l’écran avec System point out point println, comment lire un nombre à virgule avec MOOC point readDouble, comment associer une valeur numérique à un nom avec un signe égal, prixTotal égal poids fois 2,6.

Donc, on peut déjà écrire des programmes. C’est déjà un bon résultat mais on n’est pas très heureux parce que ces programmes sont tristes.

Toujours des nombres à virgule, toujours les mêmes instructions dans le même ordre, cela nous fera des choses très mécaniques. Donc, pour aller plus loin, il faut introduire plus de variétés, et c’est le but de notre semaine, d’introduire de nouvelles choses.

Plus de variétés, cela veut dire d’autres types de données avec par exemple des nombres sans virgule, des caractères, des messages, d’autres opérations, d’autres instructions, et à ce niveau-là, on va introduire une instruction qui donne plus de puissance, avec des suites d’instructions différentes d’une exécution à l’autre, qui choisit la suite d’instructions selon une ou plusieurs conditions.

Donc, cette instruction, c’est le if. If  en anglais, cela veut dire si. C’est ce qu’on appelle l’instruction conditionnelle qui permet d’avoir des exécutions différentes et des instructions différentes selon le cas.

Donc notre programme, c’est les types, les opérations, l’instruction if, et puis on va essayer d’approfondir sur les instructions qu’on connaît déjà et voilà donc le programme de la semaine.

Bonjour.

Dans cette séquence, nous allons voir une des instructions les plus importantes en Java, c’est l’instruction if qui permet de faire varier les instructions exécutées lors d’une exécution d’un programme.

Un programme, on l’a vu, c’est une suite d’instructions. L’instruction if, c’est un aiguillage qui permet de choisir entre deux suites d’instructions possibles et parmi les deux, quand on a un aiguillage, il faut choisir un des côtés et on ne va que d’un seul des deux côtés. Donc c’est le cas ici, une seule des deux suites d’instructions est exécutée lors d’une exécution donnée.

Mais d’une exécution à l’autre, ce ne sera pas toujours la même voie qui sera utilisée, d’où l’intérêt d’avoir cet aiguillage.

Et le choix en question dépend d’une condition, une condition qui est un calcul Java.

Voilà un petit dessin qui illustre le if. Donc on a le début du programme avec des instructions qui se feront l’une après l’autre, et puis on arrive à un autre aiguillage ici et le choix du côté dépend de la condition.

La condition, c’est un calcul qui peut être vrai ou faux, et si c’est vrai, on va exécuter ces instructions qui sont de ce côté-là. Si la condition est fausse, ce sont les autres instructions qui sont exécutées.

Une fois la fin de la séquence d’instructions terminée, c’est la suite du programme qui continue avec d’autres instructions l’une après l’autre.

Je vais prendre comme exemple une variante de notre programme Patate dans lequel le marchand décide de faire un prix différent selon la quantité.

En dessous de 3 kg, le prix reste 2,60 €. Au-dessus de 3 kg, le prix devient 2,30 €.

Je rappelle rapidement le programme Patate et ce qui doit changer dans notre programme Patate, c’est l’endroit où intervient le prix.

Donc, voilà ce qu’il faut faire. Dans ce brouillon, le Java est écrit en bleu et ce qui est écrit en vert, c’est ce qu’il faut que je transforme progressivement en Java.

Donc, s’il y a moins de 3 kg, je prends 2,60 € sinon, c’est-à-dire à plus de 3 kg, le prix c’est 2,30 €.

Alors, moins de 3 kg, comment cela va s’écrire en Java ? C’est facile : poids, qui est le nom donné au poids du sac, inférieur à 3 point 0. 3 point 0 parce que poids est un nombre à virgule, je vais le comparer à un autre nombre à virgule. Et là le si, sinon c’est la nouvelle instruction, le if qui s’écrit comme ceci donc en Java.

If, la condition entre parenthèses, et puis ici les instructions qu’il faudra exécuter l’une après l’autre si la condition est vraie. En dessous, les instructions à exécuter l’une après l’autre si la condition est fausse.

Donc, la condition s’écrit entre parenthèses, je l’ai déjà dit deux fois. Il s’agit d’un calcul ayant un résultat vrai ou faux. En Java et en anglais, vrai se dit true, faux se dit false. Ce résultat, appartient à un type Java qu’on appelle le type boolean qu’on verra dans un autre support.

Les suites d’instructions n’ont pas nécessairement la même longueur. On peut avoir un nombre d’instructions différent et on met juste les instructions qu’il nous faut dans un cas et dans l’autre.

Au niveau de la présentation, il y a quelque chose d’important, c’est que pour bien voir ce qui est à l’intérieur du if, on décale les deux listes d’instructions vers la droite.

Donc ici, vous voyez, toutes mes instructions sont alignées ici sauf celles qui sont à l’intérieur du if qui, elles, ont leur propre alignement décalé. Et pour décaler là, on introduit deux, trois… Ici, je pense que j’ai trois espaces, trois ou quatre, je dirais. C’est le bon nombre. Donc c’est le travail du programmeur de s’assurer que la présentation est correcte.

Le programme, il y a deux temps : le moment où je l’écris et là, quand j’écris mon programme, je ne sais pas si la condition sera vraie ou fausse. Je ne sais pas s’il y a plus ou moins de 3 kg de pommes de terre dans le sac qui n’existe pas encore.

Au moment où le programme s’exécute, la valeur de la condition est connue. Effectivement, j’ai fait rentrer du clavier le poids du sac de patates, donc la valeur est connue. Et d’une exécution à l’autre, ce qui est rentré au clavier peut être différent, plus petit ou plus grand que 3 point 0.

Avec cette instruction, avec l’instruction if, on commence à avoir des programmes qui n’exécuteront pas les mêmes instructions à chaque fois. Selon les if  et selon les conditions, les instructions exécutées changent et le if c’est une instruction complexe, c’est une instruction qui contient elle-même deux suites d’instructions. Donc c’est une instruction composée et, avec cette instruction, on rentre dans le monde de la programmation proprement dite. C’est vraiment une étape importante que nous franchissons maintenant.

Bonjour.

Nous allons, dans cette vidéo, présenter les types et les valeurs. Des types, on en a déjà vu un, c’est le type des nombres à virgule. Et dans un type, on a déjà le nom du type. Donc, pour les nombres à virgule, c’est double qui est un mot-clé de Java.

On a des valeurs. Donc on a vu que les nombres à virgule s’écrivent avec un point. Donc par exemple 12 point 5, ou moins 25 point 02 sont des valeurs du type.

On a des opérations. Donc sur les nombres à virgule, les exemples d’opération, c’est les opérations arithmétiques. On a une opération de lecture au clavier, MOOC point readDouble et deux instructions pour l’écriture à l’écran, des instructions qui sont différentes par le fait qu’il y a ln ici qui fait que quand on a ln, on passe à la ligne après avoir affiché le nombre en question, voilà.

Donc, pour tous les types, on aura ces différentes choses.

Voyons donc le deuxième type, les nombres entiers, les nombres sans virgule. Le nom du type, c’est int qui est l’abréviation de l’anglais integer qui veut dire entier en anglais, quand il s’agit d’un nombre. Les valeurs, sans surprise, cela s’écrit comme vous avez l’habitude d’écrire les nombres, 138 ou moins 96. Les opérations, également, ce sont les opérations arithmétiques.

Pour la lecture au clavier MOOC point readInt, pour les écritures à l’écran, System out point print, System out point println.

Un petit warning, un petit avertissement, la division des nombres entiers, c’est une division entière, ce qu’on appelle la division euclidienne. Donc par exemple, 7 divisé par 3 en Java, c’est égal à 2. La division entière, cela consiste à chercher, à exprimer une équation de ce genre, de dire donc là c’est la division de 7 par 3. On dit que 7 est égal à 2 fois 3 plus 1. Le diviseur, c’est 2, le reste c’est 1. Et donc, l’opération qui se note avec une barre comme cela /,la division, nous calcule le diviseur qui est un entier. Il y a un autre opérateur qui s’écrit pour cent et qui calcule le reste de la division, donc 7 pourcent 3 est égal à 1.

Voyons maintenant un autre type, le type des caractères. Le nom du type, c’est char, c’est l’abréviation de l’anglais character qui veut dire caractère. Les valeurs s’écrivent avec des petites apostrophes. Donc, ‘a’ entre deux apostrophes, c’est un caractère. Un autre caractère, c’est un signe de ponctuation, par exemple le point-virgule. Ici, on ne voit rien mais ce que j’ai entre mes apostrophes, c’est un espace que j’ai tapé avec la barre d’espace de mon clavier, et ‘1’ c’est le caractère 1, pareil, qui est tapé au clavier et entre les petites apostrophes.

Sur les caractères, il n’y a pas d’opération intéressante, pas d’opération très naturelle en tout cas.

Pour la lecture au clavier, MOOC point readChar avec un C majuscule et pour l’écriture à l’écran, System out point print ; System out point println ».

Nouveau type, le type des messages. Le nom de type, c’est String. Attention, ici, il faut un S majuscule. String en anglais, cela veut dire chaîne et le terme informatique, c’est qu’on parle de chaîne de caractères.

Des exemples, donc là, cela s’écrit entre guillemets, une double quote et dans une chaîne de caractères, on peut mettre des espaces, donc ‘’Alan Turing’’ par exemple, le nom d’un grand informaticien. On a une opération intéressante sur les Strings, c’est le plus qui fait la concaténation de chaîne.

Donc, s’il faut illustrer la concaténation de chaîne, si je fais ‘’ab plus cd’’, le résultat va être une chaîne avec ‘’abcd’’. Je ne sais pas si vous pouvez lire, donc je réécris ici, c’était peut-être caché derrière moi.

L’instruction de lecture au clavier MOOC point readString, et les deux instructions d’écriture à l’écran, System out point print, System out point println.

Alors, l’opérateur plus, c’est la même notation. Le signe plus est utilisé pour plusieurs opérations différentes. On a l’addition des nombres entiers, l’addition des nombres à virgule. Dans les deux cas, ce sont des additions. On peut dire que c’est la même opération sur des types différents, mais la concaténation oui, c’est difficile de la voir comme une addition, c’est donc une opération différente. C’est juste la notation qui est la même.

Alors, un petit exercice : que sont en Java 1, ‘1’, 1.0 et ‘’1’’ écrit de cette façon. Eh bien 1 écrit comme cela, c’est un nombre entier. Ecrit entre apostrophes, c’est un caractère type char. 1 point 0, c’est un nombre à virgule, un double. ‘’1’’, entre guillemets, c’est un String, un message.

Les opérations de comparaison qui permettent de comparer deux valeurs du même type. On a d’abord l’égalité : égal égal qui permet de tester. Donc par exemple, je peux tester si 1 est égal égal à 2. La réponse, ici, c’est faux, 1 n’est pas égal à 2. Résultat false.

Cela permet de comparer deux int, deux doubles ou deux char. Mais cela ne permet pas de comparer deux String. On a la comparaison inverse. On va vérifier que deux valeurs sont différentes. Différent, cela s’écrit point d’exclamation égal et c’est une approximation avec les caractères du clavier du signe différent qu’on n’a pas sur le clavier. Comme on ne l’a pas, on a cette espèce de notation qui fait vaguement penser à ce signe.

Exemple, 1 différent de 2, 1 est effectivement différent de 2, donc le résultat de la comparaison, ici, c’est true.

Alors, attention au signe égal parce que le signe égal est utilisé pour deux choses différentes, selon qu’on l’écrit égal ou égal égal.

Le signe égal permet d’associer un nom et une valeur. On fait  nom égal un certain calcul. Le nom est à gauche du égal, la valeur qu’on veut donner est à droite.

Égal égal permet de comparer deux valeurs : calcul égal égal calcul.

Égal égal n’est pas une instruction, c’est un opérateur comme plus qui permet de calculer un résultat et ce résultat, c’est true ou false. Donc, quand écrit poids égal 1 point 7, c’est pour dire : j’associe 1 point 7 comme valeur ; je donne cette valeur ; je l’associe au nom poids. C’est une instruction et cela peut venir seul sur la ligne à condition de mettre point-virgule à la suite.

Poids égal égal 1 point 7, c’est un calcul et cela n’est pas une instruction, cela ne peut pas apparaître seule sur une ligne, cela peut apparaître par exemple dans un if. Cela peut être la condition d’un if.

On a d’autres opérateurs pour faire des comparaisons, c’est les comparaisons d’ordre qui se notent inférieur, supérieur avec les signes qu’on a au clavier.

Par exemple 1 inférieur à 5, oui, c’est vrai. Et pour inférieur ou égal, on note de cette façon, ce qu’en mathématiques, on note respectivement de cette façon. Comme on n’a pas ces signes là au clavier, on utilise les deux signes successivement.

1 inférieur ou égal 1 ? Oui, c’est vrai.

Donc, on a comme cela un certain nombre d’opérateurs de comparaison, le égal, le différent, les comparaisons d’ordre.

Généralement, cela ne s’utilise pas comme dans mes exemples pour comparer 1 et 2. 1 et 2, je sais bien que 1 c’est différent de 2, que ce n’est pas égal, que 1 est plus petit que 2. Donc, généralement, on compare au moins un élément inconnu au moment où on écrit le programme, éventuellement deux. On peut comparer deux éléments qui sont tous les deux inconnus.

Donc par exemple poids, c’est un nom que je donne à quelque chose que je ne connais pas. Je compare ce poids que je ne connais pas encore à 5 point 0.

C’est particulièrement utile pour l’instruction if pour faire les comparaisons.

Il existe en Java un autre type que je n’ai pas encore présenté, c’est le type des résultats de comparaison. Ce type, il a comme nom boolean, comme valeurs true et false qui s’écrivent comme cela en toutes lettres. En Java, on écrit true et false comme en anglais.

Comme opérateur sur ce type-là, on a des opérateurs logiques qui sont le et logique et le ou logique.

Le et logique, il est noté comme cela avec deux caractères (&&). Ce caractère s’appelle esperluette. Je ne sais pas si vous connaissez ce nom, c’est le et commercial aussi, on peut dire.

Le ou, c’est la barre verticale deux fois (||), la barre verticale qui s’obtient sur le clavier avec la touche AltGr. La lecture MOOC point readBoolean la lecture au clavier si jamais on veut dire un boolean au clavier, et l’écriture à l’écran avec System out point print et System out point println.

Donc ce type, on peut l’utiliser, notamment pour donner un nom au résultat d’une comparaison et cela peut être utile si on veut utiliser ce résultat plusieurs fois dans notre programme. Donc, on peut déclarer un nom. Ici, je déclare le nom poidsEstLourd et je dis que c’est un boolean, donc c’est un nom qui sera associé à une des deux valeurs true ou false.

Donc, si je reprends ce morceau de programme, je commence par déclarer un poids pour un nombre à virgule que je ne connais pas. Je lis ce nombre à virgule au clavier. Je déclare un nouveau nom poidsEstLourd et je donne à poidsEstLourd le résultat de la comparaison entre le nombre à virgule que j’ai rentré, est-ce qu’il est plus grand ou égal à 5 virgule 0.

Si le nombre entré est plus grand, j’aurais true comme résultat du calcul et ce sera true qui sera associé à poidsEstLourd.

En revanche, si le poids est un nombre plus petit que 5 point 0, ce sera false qui sera associé à poidsEstLourd.

Voilà, sur cet exemple de variables et de types booléens, je termine ce tour d’horizon des types et des valeurs.

S2.4

Dans cette vidéo, nous allons revenir sur les instructions que nous avons déjà vues pour approfondir les connaissances sur ces instructions. Donc le titre c’est Instructions et variables.

On va commencer avec la notion de variable. Une variable, c’est un nom donné dans un programme à une donnée ou au résultat d’un calcul. Donc, si vous vous rappelez de notre programme Patates : poids, prixTotal sont des variables. C’est une donnée ou un résultat inconnu à l’écriture du programme et c’est pour cela qu’on leur donne un nom. C’est connu pour une exécution donnée du programme et, au moment où on écrit le programme, on ne connaît pas la valeur exacte mais on connaît le type de la donnée ou du résultat. Poids, par exemple, est un nombre à virgule. Donc, on a le type double et on a cette déclaration où on donne le nom du type suivi du nom de la variable. Double poids, et cette déclaration, c’est une instruction. On trouve cela sur une ligne Java et cela pourrait en fait créer la variable. A partir du moment où on l’a déclarée, on peut l’utiliser mais pas avant. Et l’effet, concrètement, c’est de réserver une place en mémoire pour stocker la donnée. Une fois la variable déclarée, on peut l’utiliser. Et l’utiliser, cela veut dire déjà lui donner une valeur. Et cela, cela se fait avec le signe égal. Poids égal 1 point 35 : c’est une instruction et le nom de cette instruction, l’instruction qui est notée avec le égal : son nom c’est une affectation. On affecte une valeur à la variable. Cela s’écrit comme cela : nom égal calcul. A gauche du égal, c’est le nom d’une variable déjà déclarée. A droite du égal, c’est n’importe quel calcul du bon type. Le bon type, cela veut dire que le calcul doit être du même type que celui qu’on a déclaré pour la variable. Ici, poids a été déclaré comme un double. Il faut que le calcul ici soit un calcul qui donne un nombre à virgule.

Voyons les différents types de calculs que l’on peut avoir.

On peut avoir une valeur directement : 12,5 c’est un calcul très simple qui permet de calculer 12,5.

Notre sorte de calcul, c’est le nom d’une variable.

Donc par exemple, poids c’est un calcul qui permet de savoir quelle est la valeur associée à ce nom poids, donc on va chercher cette valeur dans la mémoire.

Autre type de calcul, une méthode. Par exemple, MOOC point readDouble, cela c’est un appel de méthode et cela nous renvoie à un nombre à virgule, donc pour Java, c’est un calcul.

Et on a enfin les calculs avec opérateur, l’opérateur étoile ici, la multiplication et, de chaque côté, il faut que ce soit un autre calcul. Donc, poids, c’est un calcul tel qu’on l’a vu là, 2 virgule 6, c’est un calcul tel qu’on l’a vu là.

Tous ces calculs peuvent être utilisés dans une affectation et voilà ce que cela va donner. Ici l’affectation, poids égal autre. Là, on voit, on a deux noms de variables et là, il faut bien distinguer ce qui est à gauche, c’est la variable dans laquelle on veut mettre le résultat du calcul, et autre qui est un calcul.

Poids égal MOOC.readDouble, donc cela on le fait assez souvent et cela on l’a vu aussi dans notre programme.

Les noms de variables maintenant. Un nom, il est composé de lettres, chiffres et le signe souligné. Il ne faut pas mettre dans un nom un espace ni un signe de ponctuation. C’est pour cela par exemple prixTotal, on l’écrit prixTotal sans mettre d’espace entre prix et Total.

Un nom de variable donc, il peut apparaître à gauche ou à droite du égal. A gauche, c’est le nom de la variable qui prend une nouvelle valeur, et à droite, dans un calcul, le nom de la variable désigne ce qu’il y a en mémoire dans l’espace réservé pour cette variable. Et le contenu, c’est une valeur, un nombre, un caractère, un booléen, un message selon le type.

Prenons un exemple : la première instruction ici, double poids, va nous réserver en mémoire un espace pour stocker un nombre à virgule. Le deuxième, idem. Ensuite, l’affectation, ici, il y a le calcul 1 point 5 qui permet de calculer la valeur 1 point 5 et cela va être stocké dans la mémoire, dans l’espace associé à poids ici.

Ici, un autre calcul un peu plus compliqué, on va multiplier poids par 2 virgule 6. Poids, cela désigne une valeur stockée ici, 1 point 5 fois 2 point 6. Résultat 3 point 9.

3 point 9 stocké dans prixTotal, c’est pour cela qu’on le retrouve ici.

Si on fait deux affectations successives dans la même variable : ici donc une variable qui est de type nombre entier, truc : truc égal 12 : 12 est un calcul dont le résultat est 12, donc 12 est stocké ici.

Deuxième affectation : truc égal 5 et bien 5 est mis dans truc. La valeur 12, l’ancienne valeur de la variable est perdue, ce qui n’est pas grave si on n’en a plus besoin.

On peut avoir la même variable des deux côtés du égal. On peut écrire quelque chose comme ceci. On déclare une variable nombre. On lui affecte une valeur et ensuite, on refait une affectation où on voit qu’on a nombre à gauche, nombre à droite.

Pour comprendre tout ce qui se passe dans une affectation de ce genre, il faut savoir que l’exécution commence par le calcul, le calcul qui est à droite. Donc cela commence par nombre plus 1.

Nombre plus 1, à ce moment-là, nombre est associé à la valeur 0, donc 0 plus 1 : 1. Et le résultat du calcul 1 est mis dans la variable nombre. Donc c’est comme cela que cela se passe.

On peut déclarer et affecter un résultat, un nombre dans une seule et même instruction. Donc là par exemple, je fais double prixKilo égal 2 virgule 6. Cela me déclare la variable prixKilo et cela lui met comme valeur, cela stocke la valeur 2 virgule 6 dans l’espace mémoire correspondante. Donc, c’est équivalent à ce qu’on a fait jusqu’ici en deux instructions, une déclaration suivie d’une affectation. C’est juste un raccourci d’écriture.

Passons maintenant à un autre type d’instructions que sont les méthodes. Deux exemples de méthode qu’on a vus, MOOC point readDouble et System out point print.

Une méthode, c’est comme un petit programme qui a ses entrées et éventuellement une sortie. Les entrées, c’est ce qu’on appelle les paramètres de la méthode. Il peut y en avoir 0, 1, 2, etc. .

L’appel de méthode, c’est la demande de l’exécution du petit programme en précisant les entrées. Donc, les méthodes qu’on utilise, nous, ce sont des petits programmes que nous n’écrivons pas, qu’on utilise seulement. Donc, ce qu’on va avoir dans nos programmes, ce sont des appels de méthodes. Et les entrées du programme, on ne les donne pas au clavier, on les donne écrites dans le programme entre parenthèses.

Par exemple, dans System out point println, on fait rentrer dans le programme System out point println dont le but est d’afficher à l’écran quelque chose, on lui donne entre les parenthèses son entrée : 12 point 3.

Autre exemple d’appel, on a vu une instruction comme cela dans notre premier programme MOOC point readDouble. Ici, entre parenthèses, on n’a rien. Cela veut dire que notre programme n’a pas besoin d’entrée, readDouble n’a pas besoin d’entrée pour nous renvoyer un nombre à virgule. Il a besoin d’une entrée au clavier mais pas d’une entrée venant du programme. Il n’a pas de paramètre.

Donc, la forme générale d’une méthode, c’est le nom de la méthode, avec généralement un nom, un point, un autre nom. Cela, c’est ce que j’appelle le nom de la méthode, une parenthèse et puis les différents paramètres séparés par des virgules. Le nombre de paramètres pour une méthode donnée est fixe.

A l’appel de la méthode, il faut donc donner les paramètres attendus par cette méthode et chaque paramètre est donné sous la forme d’un calcul. Et on a déjà vu pour l’affectation qu’il y a plusieurs sortes de calculs. On a quatre sortes de calculs : les valeurs, les noms de variables, les méthodes et un calcul avec opérateur. Et on peut utiliser tout cela comme calcul pour donner des entrées à un appel de méthode. Par exemple, j’appelle la méthode println avec une valeur directement. Je peux aussi lui donner le nom d’une variable, on a déjà fait cela, quand on donne le nom d’une variable, c’est pour afficher le contenu de la variable.

Chose plus surprenante peut-être, je peux donner un appel de méthode comme calcul pour calculer ce que je veux afficher à l’écran. Donc par exemple, ici, je fais MOOC point readDouble, cela va lire au clavier le nombre et cette méthode-là va afficher ce qui est lu au clavier sans que cela passe dans une variable ou autrement dans le programme.

Donc dans cette histoire, j’ai deux appels de méthode : l’appel de méthode qui est utilisé comme calcul, pour calculer le paramètre de la méthode System out point println. Et puis, je peux aussi faire un calcul plus habituel avec un opérateur.

Certaines méthodes ont une sortie, un résultat qui est lui-même une valeur. Par exemple, MOOC point readDouble renvoie à un résultat qui est un nombre à virgule. Donc MOOC point readDouble c’est un calcul.

D’autres méthodes ne renvoient pas de résultat. Elles n’ont pas de sortie. Par exemple, System out point println : cela ne renvoie rien. C’est une méthode, si j’essaie de la représenter, elle a une entrée, ici un nombre à virgule, mais elle n’a pas de sortie. Ce sont des instructions et les effets des instructions sans sortie, ce sont soit un affichage à l’écran, soit une modification de la mémoire. Donc, bien qu’elles n’affichent rien, ces méthodes sont utiles. Il n’y a aucun doute sur le fait que le System out point println est utile.

Donc, les méthodes qui renvoient à un résultat sont utilisables en tant que calcul, qu’on peut utiliser à droite d’une affectation ou comme paramètre d’une autre méthode avec un opérateur pour calculer un résultat. Celles qui ne renvoient pas de résultat sont des instructions qu’on voit sur une ligne séparée.

La librairie MOOC, c’est un ensemble de méthodes qu’on utilise pour notre cours. Ce qu’il y a dans MOOC, c’est différent, par exemple de System out point println.

System out point println c’est du Java standard, vous pouvez l’utiliser sur n’importe quel ordinateur qui a un Java.

En revanche, ce qu’il y a dans MOOC, c’est spécifique à notre cours et c’est une bibliothèque qui contient un certain nombre de méthodes que j’ai écrites pour vous faciliter la vie. Mais la contrepartie de cette facilitation, c’est qu’il faut, pour pouvoir utiliser la librairie MOOC, télécharger un fichier spécifique qui s’appelle MOOC point Java, avant de pouvoir l’utiliser sur un ordinateur standard. Cette librairie MOOC vous est donnée en annexe de ce document.

Donc, voilà. On a fait un tour d’horizon des différentes instructions que nous avons déjà vues la semaine précédente.

Je vous propose un programme avec l’utilisation de l’instruction  if, qui est une variante du programme Patate. Dans cette variante, le marchand va consentir un prix préférentiel à ses bons clients. Les bons clients, c’est ceux qu’il voit toutes les semaines. Donc, c’est le marchand qui reconnaît à leur tête ses bons clients.

Par rapport à notre programme, on va avoir toujours le poids, qui va être une entrée, mais on va avoir une deuxième entrée, qui est l’information : est-ce un bon client ou pas ? Et puis la sortie, cela va être le prix à payer. Et le prix, en interne, on aura deux prix : un prix pour les clients lambda et un prix pour les bons clients.

Une fois qu’on a vu ce schéma d’interaction, ces entrées du programme, la question se pose : quels sont les types ? Le poids, on sait que c’est un nombre à virgule. Mais bon client, qu’est-ce que cela va être comme type Java ? Cela pourrait être, on peut se dire bon client, c’est une information en vrai-faux, cela pourrait être un booléen. Mais ce ne sera pas très pratique pour le marchand de rentrer true si c’est un bon client, false si c’est un mauvais client. Donc on va plutôt utiliser un code, dans lequel on va utiliser un caractère pour signifier que c’est un bon client, et donc je vous propose d’utiliser O pour oui et le caractère N pour non. O pour oui, c’est un bon client, N : non, ce n’est pas un bon client. Donc le type qu’on va utiliser ici, c’est char.

Je vais basculer en Drjava et je pars du programme Patate. Je vais le changer et donc je vais lui donner un nouveau nom et le sauver dans un nouveau fichier pour ne pas perdre l’ancien programme. Et maintenant, je vais le modifier. J’ai besoin d’une deuxième entrée après le poids, j’ai besoin d’un char qui me dit si c’est un bon client ou pas. Et donc la deuxième entrée, je vais la faire après la première : System point out point println. Vous voyez qu’ici, j’ai l’entrée du poids et la lecture du double. L’entrée de l’information, si c’est un bon client, se fera de la même façon, avec d’abord un message : est-ce un bon client ? Puis bonclient  (qu’il faut que je tape correctement) égal MOOC point readChar. Je vais lui préciser que j’attends une réponse O ou N. Puis, c’est ici que je vais avoir besoin d’un aiguillage. Si c’est un bon client, je vais l’écrire if bon client égal égal O pour oui, dans ce cas-là, prixTotal, cela va être poids fois 2 point 30. Sinon, le prixTotal, ce sera comme avant, 2 60.

Voilà. Donc, j’ai mon if avec ces deux côtés, cet aiguillage avec les deux côtés selon que le client est un bon client ou pas. L’affichage du résultat ne change pas, mon programme est terminé. Je le sauve et je le compile. Voilà, la compilation est terminée, je vais maintenant exécuter mon programme. Je vais prendre 1,5 kg. Est-ce un bon client ? Oui. Voilà le prix pour un bon client.

Une remarque à ce niveau-là, c’est qu’on voit que le résultat du calcul là n’est pas exact. Quand je fais : 2 virgule 3 fois 1 point 5, le résultat devrait être 3 virgule 45. Et vous voyez que ce n’est pas tout à fait 3 virgule 45 mais presque. Pourquoi cela ? Parce que les ordinateurs, pour les nombres avec virgule, font des calculs approximatifs, ce n’est pas le calcul exact. C’est surprenant, mais c’est comme que cela marche. C’est comme cela que c’est en Java et dans la plupart des langages de programmation, c’est pareil. Donc il faut le savoir. C’est sûr qu’en termes d‘affichage, on préfèrerait là arrondir aux 2 chiffres après la virgule, mais ce n’est pas facile à faire. Donc, c’est trop difficile pour nous débutants, on n’insiste pas là-dessus. Le fait que les calculs soient approximatifs, ce n’est pas dramatique. Là, vous voyez, c’est vraiment une fraction minime de centimes. Simplement, il faut le savoir. C’est important de le savoir et de le noter.

Donc, 3,45 si c’est un bon client. Maintenant, on va recommencer le programme, 1,5 kg si ce n’est pas un bon client : là, c’est à peu près 3,90 le prix.

Voilà. Donc ces calculs approximatifs, cela peut quand même surprendre les clients, c’est sûr, qu’on leur demande des fractions de centime… C’est un défaut de notre programme, qu’on n’est pas en mesure de corriger pour le moment. Voilà.

Je vous propose un programme avec l’utilisation de l’instruction  if, qui est une variante du programme Patate. Dans cette variante, le marchand va consentir un prix préférentiel à ses bons clients. Les bons clients, c’est ceux qu’il voit toutes les semaines. Donc, c’est le marchand qui reconnaît à leur tête ses bons clients.

Par rapport à notre programme, on va avoir toujours le poids, qui va être une entrée, mais on va avoir une deuxième entrée, qui est l’information : est-ce un bon client ou pas ? Et puis la sortie, cela va être le prix à payer. Et le prix, en interne, on aura deux prix : un prix pour les clients lambda et un prix pour les bons clients.

Une fois qu’on a vu ce schéma d’interaction, ces entrées du programme, la question se pose : quels sont les types ? Le poids, on sait que c’est un nombre à virgule. Mais bon client, qu’est-ce que cela va être comme type Java ? Cela pourrait être, on peut se dire bon client, c’est une information en vrai-faux, cela pourrait être un booléen. Mais ce ne sera pas très pratique pour le marchand de rentrer true si c’est un bon client, false si c’est un mauvais client. Donc on va plutôt utiliser un code, dans lequel on va utiliser un caractère pour signifier que c’est un bon client, et donc je vous propose d’utiliser O pour oui et le caractère N pour non. O pour oui, c’est un bon client, N : non, ce n’est pas un bon client. Donc le type qu’on va utiliser ici, c’est char.

Je vais basculer en Drjava et je pars du programme Patate. Je vais le changer et donc je vais lui donner un nouveau nom et le sauver dans un nouveau fichier pour ne pas perdre l’ancien programme. Et maintenant, je vais le modifier. J’ai besoin d’une deuxième entrée après le poids, j’ai besoin d’un char qui me dit si c’est un bon client ou pas. Et donc la deuxième entrée, je vais la faire après la première : System point out point println. Vous voyez qu’ici, j’ai l’entrée du poids et la lecture du double. L’entrée de l’information, si c’est un bon client, se fera de la même façon, avec d’abord un message : est-ce un bon client ? Puis bonclient  (qu’il faut que je tape correctement) égal MOOC point readChar. Je vais lui préciser que j’attends une réponse O ou N. Puis, c’est ici que je vais avoir besoin d’un aiguillage. Si c’est un bon client, je vais l’écrire if bon client égal égal O pour oui, dans ce cas-là, prixTotal, cela va être poids fois 2 point 30. Sinon, le prixTotal, ce sera comme avant, 2 60.

Voilà. Donc, j’ai mon if avec ces deux côtés, cet aiguillage avec les deux côtés selon que le client est un bon client ou pas. L’affichage du résultat ne change pas, mon programme est terminé. Je le sauve et je le compile. Voilà, la compilation est terminée, je vais maintenant exécuter mon programme. Je vais prendre 1,5 kg. Est-ce un bon client ? Oui. Voilà le prix pour un bon client.

Une remarque à ce niveau-là, c’est qu’on voit que le résultat du calcul là n’est pas exact. Quand je fais : 2 virgule 3 fois 1 point 5, le résultat devrait être 3 virgule 45. Et vous voyez que ce n’est pas tout à fait 3 virgule 45 mais presque. Pourquoi cela ? Parce que les ordinateurs, pour les nombres avec virgule, font des calculs approximatifs, ce n’est pas le calcul exact. C’est surprenant, mais c’est comme que cela marche. C’est comme cela que c’est en Java et dans la plupart des langages de programmation, c’est pareil. Donc il faut le savoir. C’est sûr qu’en termes d‘affichage, on préfèrerait là arrondir aux 2 chiffres après la virgule, mais ce n’est pas facile à faire. Donc, c’est trop difficile pour nous débutants, on n’insiste pas là-dessus. Le fait que les calculs soient approximatifs, ce n’est pas dramatique. Là, vous voyez, c’est vraiment une fraction minime de centimes. Simplement, il faut le savoir. C’est important de le savoir et de le noter.

Donc, 3,45 si c’est un bon client. Maintenant, on va recommencer le programme, 1,5 kg si ce n’est pas un bon client : là, c’est à peu près 3,90 le prix.

Voilà. Donc ces calculs approximatifs, cela peut quand même surprendre les clients, c’est sûr, qu’on leur demande des fractions de centime… C’est un défaut de notre programme, qu’on n’est pas en mesure de corriger pour le moment.

Resume

Je vous présente Simon CHAUVIN, qui est doctorant au CNAM et qui fait un doctorat sur les jeux vidéos, qui auparavant a été formé à l’Ecole nationale du jeu et des médias interactifs et qui est venu nous présenté un jeu fait en java appelé Minecraft.

Simon CHAUVIN : tout à fait, Minecraft on le voit, c’est un jeu à la première personne où le joueur en fait a la possibilité de se déplacer dans un monde 3D entièrement généré procéduralement, c’est-à-dire qu’à chaque partie, c’est un monde différent dans lequel le joueur se trouve. Dans ce monde là, le jour va avoir la possibilité de construire des choses, il va avoir la possibilité de récupérer des objets, de récupérer des blocs de terre ou pierres, de métal et qui vont lui permettre de reconstruire tout un tas de choses. Pour cela il va devoir essayer de créer de nouveaux outils, et donc on va se retrouver aussi dans un monde où il y a beaucoup d’animaux, des monstres où il y a de la larve, où il y a de l’eau, et en fait tous ces éléments-là vont nous permettre de reconstruire un monde différent, un monde qui a des joueurs qui vont s’amuser à recréer des villes entières, à recréer leur maison de leur enfance, donc c’est un jeu qui permet beaucoup de possibilités.

François BARTHELEMY : oui, l’éditeur de ce jeu revendique plus de 100 000 000 d’utilisateurs, plus de 15 000 000 d’exemplaires vendus, donc un gros succès que certains d’entre vous reconnaissent certainement, est-ce que tu peux nous retracer rapidement l’histoire de ce jeu ?

Simon CHAUVIN : oui, tout-à-fait. Minecraft c’est un jeu qui a été réalisé par Marcus PERSSON qui était tout seul, en fait qui a fait ce jeu tout seul, il a très vite très bien fonctionné, c’est-à-dire qu’il y a beaucoup de joueurs qui ont créé une communauté autour du jeu, avec beaucoup de serveurs sur lesquels on pouvait se connecter et jouer en multi-joueurs, donc il y a des joueurs qui se sont mis à faire toutes sortes de choses, que ce soit recréer la Tour Eiffel dans Minecraft ou recréer l’Enterprise de Starstrek dans le jeu, donc on a toute une communauté de joueurs qui a décidé d’utiliser Minecraft de plein de manières différentes, de plein de manières qui n’ont pas été forcément prévues par le créateur à la base.

François BARTHELEMY : et à un moment cette communauté est allée plus loin en voulant s’approprier le jeu lui-même, le fonctionnement du jeu.

Simon CHAUVIN : tout à fait, effectivement, le programme Minecraft qui est programme java qui est donc compilé en 1.jar, en fait une partie de cette communauté là a décidé de le décompiler pour essayer de retrouver le programme source et pour pouvoir fournir en fait des outils à tous les joueurs de Minecraft des outils pour modifier Minecraft, c’est-à-dire en fournissant les différents fonctionnalités de Minecraft sous forme de programme java en fait, cette communauté a permis à ces joueurs de venir rajouter des blogs, de venir rajouter des personnages de venir rajouter des règles de jeux en fait simplement qui transforment Minecraft dans sa totalité.

François BARTHELEMY : Ce processus de décompilation dont tu nous parles c’est quelque chose d’illégal, c’est du piratage.

Simon CHAUVIN : c’est quelque chose d’illégal, effectivement, oui.

François BARTHELEMY : mais ce qu’il y a d’intéressant dans cette expérience, c’est que ce piratage finalement a bien servi l’entreprise qui était contente que toute une communauté au plus large puisse contribuer au jeu, le rendre plus intéressant. Et l’entreprise, loin de lutter contre le piratage l’accepte voire l’encourage.

Simon CHAUVIN : oui, tout à fait, cela a permis effectivement à Minecraft de devenir célèbre et de proposer des expériences vraiment totalement différentes en fonction de ce que les joueurs vont créer à partir de ces outils de modifications du jeu, effectivement.

François BARTHELEMY : donc cette possibilité de changer le jeu tu l’utilises toi pour ta thèse, tu as créé un programme java qui vient modifier le jeu, on en a pris un petit extrait pour vous montrer que c’est du java comme vous connaissez avec une variable INT ici, avec des IF, des conditions, enfin des choses que vous pouvez comprendre, qu’est-ce qu’il fait ce programme ?

Simon CHAUVIN : alors ce petit programme qui commence à prendre de l’espace, qui n’est plus si petit que ça, mais effectivement, dans ce mode-là, dans ces nouvelles fonctionnalités que j’apporte à Minecraft, l’idée c’est de venir enrichir la narration que Minecraft ne propose pas. Et donc de venir proposer aux joueurs tout un tas de façons de modifier son environnement, de rajouter des personnages qui vont avoir des buts particuliers peut-être d’embêter le joueur ou au contraire de l’aider.

François BARTHELEMY : merci. Est-ce que tu pourrais nous dire quel est l’intérêt d’utiliser java pour développer ce jeu ?

Simon CHAUVIN : oui tout à fait, java a cette particularité déjà, il est multiplateforme donc effectivement Marcus PERSSON donc le créateur du jeu a développé son jeu une seule fois et donc il a utilisé le .jar enfin, il a vendu le jeu à tous les joueurs qui ont pu l’exécuter sur leur ordinateur qu’il soit sur Linux, MacOs ou sous Windows, donc ça c’est le premier point intéressant avec java. Et deuxièmement dans la mesure où les programmes sources de Minecraft ont été rendus "publiques" par la communauté, et donc ont permis la modification de Minecraft dans la mesure où c’était du java et que c’est un langage très répandu, très connu et qui est appris dans beaucoup d’endroits , que ce soit dans les écoles d’ingénieurs, les unités ou le CNAM évidemment. Cela a permis en fait à beaucoup de joueurs de faire des modifications de Minecraft très facilement avec un langage qu’ils connaissent bien.

François BARTHELEMY : malheureusement à la fin du MOOC vous ne pourrez pas vous lancer dans cette activité de développement de programme, il vous manquera pour rajouter des choses dans Minecraft des connaissances telles que la programmation objet notamment, mais si vous continuez à vous former peut-être d’ici quelques années vous aussi vous pourrez ajouter des éléments nouveaux dans des jeux vidéos. Simon merci, peut-être pour terminer une petite question sur travailler dans le jeu. Est- ce que quand on travaille dans le jeu on joue beaucoup ?

Simon CHAUVIN : on joue, pas forcément beaucoup, des fois on est amené à devoir jouer pour tester certaines choses de nos propres jeux, mais c’est souvent pas forcément très marrant au début parce qu’il y a beaucoup de bugs, beaucoup de choses à corriger donc c’est surtout du travail.

François BARTHELEMY : plus une activité de programmeur que de joueur.

Simon CHAUVIN : oui, tout à fait.

François BARTHELEMY : bien merci.

Simon CHAUVIN : merci.

SEMAINE 3

Bonjour.

Nous allons voir l’instruction la plus compliquée qui existe en Java et qui permet d’exécuter plusieurs fois certaines instructions. Le titre de la semaine, c’est :« Ça va durer encore longtemps ?», car grâce à cette instruction, nos programmes peuvent durer beaucoup plus longtemps que quand ils exécutent une fois chaque instruction.

Un petit point sur ce qu’on a déjà vu : on a déjà vu qu’un programme est une suite d’instructions qui s’exécutent l’une après l’autre, sauf quand on a un if  qui est comme un aiguillage avec deux voies possibles et on choisit la voie selon la valeur d’une condition.

Pour certaines applications, on voudrait exécuter les mêmes instructions plusieurs fois. Par exemple, le marchand de patates, il voudrait ne lancer qu’une fois le programme pour calculer les prix de tous les sacs de patates des clients de sa matinée. Il ne voudrait pas à chaque fois avoir à relancer le programme.

Un autre exemple, c’est pour effacer l’écran. Il n’existe pas d’instruction pour effacer l’écran. On a juste des instructions pour afficher des choses à l’écran, qui sont System out point print et System out point println. Si on veut avoir un effet d’avoir un écran qui n’affiche plus rien, il faut afficher plusieurs fois une ligne blanche. Et en affichant suffisamment de fois une ligne blanche, on n’aura plus rien à l’écran. Donc pour faire cela, on va introduire une nouvelle instruction, qui permet d’exécuter plusieurs fois la même instruction, et permet de compter combien de fois on l’a exécutée. C’est l’instruction for. Le programme de la semaine, c’est donc cette instruction for.

Une autre nouveauté, c’est qu’on va regarder comment transcrire une exécution d’un programme, et puis on va faire de l’approfondissement sur des choses qu’on a déjà vues, le if, et les conditions. Voilà.

Bonjour.

Nous allons voir la boucle for qui est l’instruction la plus compliquée qu’on verra au cours de ce cours et qui est une des instructions les plus compliquées qui existent en Java. La boucle for est utilisée pour exécuter plusieurs fois les mêmes instructions et chaque exécution de ces instructions est appelée un tour de boucle. Et le principe de la boucle for, c’est d’utiliser une numérotation des tours de boucle. C’est-à-dire pour chaque tour de boucle, on a un numéro et ce numéro de tour de boucle est stocké dans une variable qui est une variable Java tout à fait standard.

Voici une boucle for, on connait donc l’instruction System out print où on voit que là cela va afficher le nombre 75. Ici, cela va afficher un espace et passer à la ligne. Donc, c’est des instructions qu’on comprend. Mais pour l’instant, je veux vous présenter les différentes parties de la boucle for. Il y a quatre morceaux dans une boucle for et on va voir successivement ces quatre morceaux.

Le premier morceau, en fait le dernier dans le texte mais le premier dont je vais vous parler, c’est ce qu’on appelle le corps de la boucle. Ce sont les instructions à exécuter plusieurs fois et on en met autant que l’on veut entre des accolades. Elles sont exécutées donc à chaque tour de boucle.

Ensuite, avant le corps de la boucle, il y a l’entête de la boucle qui comporte trois parties que nous allons voir successivement. La première partie, c’est l’initialisation. L’initialisation est exécutée une seule fois au début de la boucle. C’est la première instruction exécutée quand on commence la boucle et cela consiste à déclarer et initialiser la variable contenant le numéro de tour. Donc ici, j’ai choisi d’après cette variable num, c’est moi qui ai choisi le nom.

Le deuxième morceau, c’est une condition et donc c’est comme la condition dans le if. Cette condition, elle est calculée avant chaque tour de boucle et si la condition est vraie, le tour de boucle est effectué. Si la condition est fausse, on arrête, la boucle est terminée. Et généralement cette condition, c’est tester si le dernier numéro de tour qu’on avait prévu a été atteint ou dépassé.

L’incrémentation est exécutée à la fin de chaque tour de boucle, c’est-à-dire après l’exécution du corps et cela consiste à passer au numéro suivant. Le plus souvent, comme ici, cela consiste à ajouter 1 au numéro de boucle et donc passer au numéro suivant, c’est juste ajouter 1.

Revoyons maintenant en prenant en compte ces quatre morceaux quelles sont les instructions exécutées lors de l’exécution de la boucle. La première chose, c’est toujours l’initialisation. Ensuite, la condition. Si la condition est fausse, la boucle est terminée mais ici je suppose que la condition est vraie. Si la condition est vraie, on est parti sur un tour de boucle de plus. Faire le tour de boucle, c’est d’abord exécuter le corps et ensuite faire l’incrémentation, cette incrémentation qui passe au numéro suivant. Donc, un tour de boucle est terminé. Est-ce qu’on en fait un de plus ? La condition est recalculée. Si elle est vraie, on est reparti pour exécuter le corps une deuxième fois, l’incrémentation une deuxième fois.

Et on passe au troisième numéro et on revient à la condition. Est-ce que la condition est vraie ? Si elle est fausse, là je suppose qu’elle est fausse, la boucle est terminée.

Je vous propose d’étudier le premier exemple au moyen d’une démo. L’objet de cette démo, c’est d’effacer l’écran. On n’a pas d’instruction pour effacer l’écran et on n’a pas d’instruction pour revenir à la ligne du dessus. Donc en fait, on va tricher. Pour effacer l’écran, on va tout simplement afficher les lignes blanches et suffisamment pour que ce qu’on a écrit aille vers le haut et que l’on ne puisse pas de le voir sans manipuler l’ascenseur.

Donc l’instruction pour afficher une ligne blanche, on peut utiliser System point out point println avec un message qui ne contient par exemple qu’un espace. Si je fais cette instruction System out point println, cela a pour effet d’afficher une ligne blanche. Et afficher une ligne blanche, cela ne suffit pas. Il faut que j’en affiche plusieurs.

Donc, si je n’avais pas la boucle for, je ferais du copier-coller et il faudrait que je le fasse peut-être… Voilà, un certain nombre de fois. Ce n’est pas très agréable d’écrire… Même si le copier-coller nous aide, en tout cas en termes d’intérêt du programme, ce n’est pas très bien. Donc on préfère utiliser une boucle for et donc je vais avoir une variable, je commence par l’initialisation. La variable… Le premier tour, je vais le numéroter avec 1, là, c’est moi qui choisis, mais bon, c’est relativement standard et je me demande combien je vais en mettre ; bon, je vais aller jusqu’à 25. Donc, tant que num est plus petit ou égal à 25, je continuerais à afficher des lignes et l’incrémentation, cela va être d’ajouter 1 au numéro.

Comme je suis dans une boucle for, il faut que je décale ce qu’il y a à l’intérieur de mon corps. Visuellement, cela doit être décalé par rapport au for, pour dire que c’est une instruction à l’intérieur de mon instruction for.

Voilà donc, je regarde rapidement mon programme : il a l’air bien, donc on va essayer de le compiler. Il faut que je le sauve d’abord dans Effacer.Java.

La compilation s’est bien passée, j’ai un message encourageant, donc on va passer à l’exécution. Pour montrer que cela efface l’écran, il faut que je commence par afficher quelque chose sinon on risque de ne pas voir.

Je sauve, je compile : quatre erreurs, qu’est-ce qui ne va pas ? Ah oui, ce n’est pas au bon endroit, c’est cela qui ne va pas, c’est ici qu’il le faut mettre, voilà. Je sauve, je compile et cette fois ça va. Donc, on va exécuter le programme. Vous voyez, cela a effacé l’écran. Avec l’ascenseur, je peux remonter et je retrouve ce qui s’est affiché au début : bonjour et 25, voilà. Donc, on ma boucle s’est bien passée.

Je vous propose maintenant de voir quelque chose, de constater par un affichage que le numéro de tour a effectivement changé et ce numéro de tour, je peux l’utiliser dans ma boucle. Je peux utiliser cette variable que j’ai déclarée au début, qui s’appelle num et je peux en faire ce que je veux. Donc là, en l’occurrence, je vais l’afficher au début de ma ligne. Ma ligne ne sera plus vraiment blanche. C’est juste pour matérialiser lors de notre exécution le fait que le numéro de boucle change.

Allons-y, voilà.

Donc, on voit que le premier tour de boucle, c’était 1, parce qu’ici j’ai écrit 1. Le dernier, on a passé tous les nombres 1, 2, 3, 4, 5 jusqu’à 25 qui est le dernier tour de boucle et c’est bien ce que j’avais prévu, j’ai mis 25 ici, l’une inférieure ou égale à l’autre, voilà.

On va repasser sur la présentation et on va essayer de regarder maintenant en détail l’exécution de ce programme. Ce n’est pas tout à fait le même programme mais il fait la même chose : il affiche le numéro de ligne et il passe à la suite. Donc la première instruction exécutée, c’est l’initialisation. L’effet de l’initialisation, c’est de réserver : donc la déclaration de la variable nous réserve un espace pour num et on lui met la valeur 1 la valeur initiale.

Ensuite, deuxième instruction, la condition : est-ce que num est inférieur ou égal à 3 ? Num qui vaut 1 : oui, il est inférieur ou égal à 3. Donc, vous voyez ici par rapport à la démo, c’est 3 qu’on a mis ici, donc pour que mon explication ne dure pas trop longtemps, on ne va pas jusqu’à 25.

La condition est vraie, donc on exécute le corps de la boucle. Le corps de la boucle, c’est un affichage, l’affichage de num à l’écran. Alors, le voilà. num, c’est 1.

Ensuite, incrémentation num égal num plus 1. Donc, num devient 2.

Est-ce que num est inférieur ou égal à 3 ? Oui, 2 est inférieur ou égal à 3. Donc, la condition est vraie, on part sur un tour de boucle de plus. Ce tour de boucle, cela va être : afficher 2.

Afficher 2 à la ligne, mais mon écran n’est pas assez grand pour que je passe à la ligne mais il y a un ln ici, donc il faudrait afficher à la ligne.

Num égal num plus 1 ici, on a 3. Num est inférieur ou égal à 3 ? Oui, 3 est inférieur ou égal à 3. Donc, cela c’est vrai.

Je ne suis pas allé jusqu’au bout de l’exécution de la boucle mais vous voyez, cela continue. L’étape suivante, c’est d’afficher le numéro 3, donc après le 2 le 3 et, nouvelle incrémentation, ici, cela devient 4. Et on revient à la condition : est-ce que num est inférieur ou égal à 3 ? Non, 4 n’est pas inférieur ou égal à 3. La boucle est donc terminée.

Un tour de boucle est donc composé de trois choses : d’une part, la vérification de la condition qui est vraie, l’exécution du corps de la boucle qui comprend plusieurs instructions, ou une seule ou au moins une, et l’incrémentation qui consiste à passer au numéro suivant.

Avant le premier tour, on a l’initialisation.

Après le dernier tour, on a la vérification de la condition (qui est fausse).

Voilà. Avec cette instruction for, on passe dans un autre type de programme, des programmes où on répète les choses un certain nombre de fois. Cela peut durer longtemps. La boucle for, dans les exemples que j’ai pris là, c’était 3 tours ou 25 tours, mais rien n’empêche de faire une boucle pendant 100 tours, 1 000 tours, 100 000 tours, des choses qu’on fait parfois, voilà.

Voilà qui termine ce premier exposé sur les boucles en Java.

3.3

Nous ne sommes pas des ordinateurs mais nous avons besoin parfois de comprendre ce que fait l’ordinateur. Donc, nous avons besoin de retracer sur papier ce qui se passe à l’intérieur de l’ordinateur. Et je vous propose une technique qui permet de faire sur papier ce qu’on appelle la trace d’un programme, c’est-à-dire un suivi de l’exécution d’un programme.

Donc, l’exécution, c’est des instructions exécutées l’une après l’autre et l’effet d’une instruction peut être de changer le contenu de la mémoire, c’est le cas par exemple d’une déclaration. Une déclaration, cela va réserver un espace en mémoire pour une variable. Le égal, cela va changer le contenu d’une variable dans la mémoire. Cela peut être une écriture à l’écran avec System out point println ou System out point print, ou une lecture au clavier.

Pour comprendre l’exécution d’un programme, il faut prendre tout cela en compte et une façon de prendre tout cela en compte, c’est de le présenter au moyen d’un tableau sur du papier. L’idée là, c’est d’avoir un tableau avec une ligne par instruction exécutée.

En colonne 1, on mettra le numéro de ligne du programme qui s’exécute. Donc, cela veut dire qu’on numérote les lignes du programme. Les éditeurs de texte savent le faire généralement, et donc on se réfère au numéro qui est donné.

Ensuite, on va prendre la colonne 2 pour l’écran, la colonne 3 pour le clavier, donc c’est là qu’on notera ce qui s’affiche, ce qui vient du clavier.

Et ensuite, les colonnes suivantes, on en prendra une pour chaque variable. Donc, si c’est notre programme Patate, ce sera poids et prixTotal.

Evidemment, on peut aussi rajouter d’autres colonnes si on a d’autres choses intéressantes à noter. Donc il ne faut pas hésiter à ajouter une colonne si on pense que c’est utile pour la compréhension du programme.

Je vous propose le programme  Patate, donc je l’ai rappelé ici avec les numéros de ligne.

Donc, pour la trace, je vais avoir ligne, le numéro de ligne, l’écran, le clavier, le poids. L’écran, le clavier et les deux variables, donc ici on a poids, ici on a prixTotal, cela nous fait deux colonnes dans notre clavier.

L’exécution du programme commence par la ligne 1 et à partir de la ligne 1, la variable poids existe, donc je note le fait qu’elle existe ici avec un point d’interrogation. Pourquoi un point d’interrogation ? Parce qu’il n’y a pas de valeur pour le moment ou en tout cas on ne lui a pas donné de valeur. En revanche, prixTotal n’existe pas, donc, je note tout simplement qu’elle n’existe pas.

Deuxième ligne, deuxième instruction exécutée, c’est la ligne n° 2 du programme et là prixTotal existe mais on ne sait pas, cela n’a pas encore de valeur bien définie.

Ligne 3 : quel poids ? Donc, c’est un affichage, c’est dans la colonne écran que cela apparaît.

Là, une ligne à deux temps, il y a d’abord readDouble qui va lire au clavier 1 point 7 et puis là, une affectation qu’il met dans poids. Donc, dans la colonne poids, on va noter 1 point 7.

Cinquième instruction exécutée, c’est ce calcul poids fois 2 point 7, cela nous donne 4,42 qui est mise donc dans la variable prixTotal.

Un exemple un peu plus compliqué maintenant avec une boucle for.

Dans la boucle for, on voit qu’ici on a sur la même ligne, on a des choses différentes, donc on va utiliser une notation pour savoir de quoi on parle.

La première chose à exécuter, c’est l’initialisation. L’initialisation, c’est la déclaration de la variable num et on lui met la valeur 1. Vous voyez ici que dans mon tableau, j’ai trouvé intéressant de faire figurer une colonne supplémentaire qui contient la condition de la boucle.

Deuxième chose exécutée, c’est toujours sur la ligne 1, mais c’est la condition. Et quand num vaut 1, num est inférieur ou égal à 3, donc la valeur ici, c’est 3.

Comme la condition est vraie, on fait un tour de boucle de plus et on attaque le corps de la boucle. System point out point println de num. Donc, cela nous affiche le contenu de la variable num qui est 1. C’est cela qu’on voit ici à l’écran.

Et en revanche, je reviens en arrière, cet affichage n’a pas changé la variable num, donc c’est juste la valeur qui était là avant qui reste dans cette colonne-là, qui est recopiée on peut dire.

Ensuite, le tour de boucle est terminé, il reste à faire l’incrémentation. Donc, on note cela 1 point incr et l’incrémentation cela consiste à faire num égal num plus 1, donc ici num devient égal à 2.

1 point cond : est-ce que 2 est plus petit que 3 ? Oui, donc, un tour de boucle de plus avec le corps de la boucle qui consiste à afficher le 2 et l’incrémentation : num vaut 3.

Est-ce que 3 est plus petit ou égal à 3 ? Oui, donc un tour de boucle de plus, à nouveau la ligne 2 est exécutée et cela consiste à afficher le 3 à l’écran. L’incrémentation : num vaut 4.

Ensuite, instruction suivante, la condition : est-ce que 4 est plus petit ou égal à 3 ? Non. La boucle est terminée et c’est tout ce qu’on avait dans le programme.

Donc, pour récapituler, si je mets la trace d’exécution complète, voici ce que cela donne et cela récapitule tout ce qui s’est passé, tout ce qui a été exécuté. On voit le nombre d’instructions qui a été exécuté, on voit que la ligne 2 a été exécutée trois fois. On voit qu’il y a eu trois tours de boucle que je vais noter ici. Donc cette trace nous donne plein d’informations et explique ce qui se passe à différents moments de l’exécution du programme.

C’est intéressant d’avoir cette information et cela permet de comprendre l’état de la mémoire et de l’écran. Cela permet une vérification et notamment cela permet de comprendre s’il y a un comportement qu’on ne comprend pas, s’il y a un bug. On ne comprend pas mais on peut faire la trace d’exécution, ou on peut utiliser cela aussi pour montrer à quelqu’un ce qui se passe lors de l’exécution du programme.

3.4

Bien. Nous avons déjà vu des si comme des aiguillages à deux branches. Aujourd’hui, nous allons aller plus loin avec les si en voyant des aiguillages à plusieurs branches.

Donc, un petit rappel avant d’attaquer les nouveautés, c’est qu’on a vu des si qui ressemblent à cela, avec deux séquences d’instructions ou deux voies si vous voulez, deux chemins possibles et qui sont conditionnés par une condition. Si la condition est vraie, on prend un chemin. Si la condition est fausse, on prend un autre chemin.

Donc, on a vu un aiguillage comme cela où on a le choix entre deux côtés, et après on reprend.

Parmi les choses nouvelles qu’on va voir, c’est un aiguillage où on n’a pas d’instructions d’un des côtés et les instructions, on va les garder du côté où la condition est vraie. Et du côté où la condition est fausse, on passe directement ici à la suite du programme sans rien exécuter entre-temps.

Ensuite, on verra un aiguillage à trois voies où on a trois séquences d’instructions qui peuvent avoir des nombres d’instructions différents à chaque fois, cela peut être différent ici, et on a le choix entre trois possibilités.

Et ensuite, on verra un cas encore plus général où on a n voies, n étant 3, 4, 5 ou ce que vous voulez.

Commençons par l’aiguillage sans instructions et l’exemple, c’est le calcul d’une réduction.

On prend notre programme Patate et on se dit si l’acheteur prend plus de 3 kg, il a une réduction de 10 %. Et donc, là dans notre calcul, on va avoir cette structure. On commence par faire le calcul du prix. Le total, c’est ce qui est dans nos précédentes moutures du programme Patate, qui s’appelait prixTotal. On commence par calculer ce total et après, si le poids est supérieur à 3 kg, alors on applique la réduction. Appliquer la réduction, c’est ce qui est écrit ici. Donc, ce qui apparaît en bleu ici, c’est du Java, ce qui est en noir, c’est du pas encore Java.

Le si alors sinon le total est inchangé, ce n’est pas du Java. Ici en revanche, c’est le calcul de la réduction. On fait le total multiplié par 90 et diviser par 100, cela nous fait la réduction de 10 %.

Et sinon, le total est inchangé, sinon il n’y a rien à faire. En Java, cela va s’écrire comme cela, avec un if et pas de else. Il faut quand même l’accolade fermante, l’accolade qui ferme cette accolade là mais on a un if sans avoir de else et c’est comme cela qu’on fait les aiguillages avec un côté sans instructions.

Voyons maintenant un aiguillage à trois branches. Si on a le choix entre trois branches, une condition vraie ou fausse ne suffit pas. Donc en Java, il faut deux conditions qui chacune peuvent être vraies ou fausses. Les trois branches : la branche 1… (Donc a deux conditions que je vais appeler condition 1, condition 2). Donc la branche 1, on la prendra si la condition 1 est vraie, la branche 2 si la condition 1 est fausse mais la condition 2 est vraie, et la branche 3 c’est si les deux conditions sont fausses.

Donc, pour la branche 1, on ne regarde qu’une des deux conditions. Pour la branche 1, on ne regarde même pas la deuxième condition. Notre exemple, on va distinguer trois prix. L’exemple qu’on avait pris la semaine dernière pour introduire le « if », c’est s’il y a plus ou moins de 3 ou 5 kg. Ici, on va prendre des prix différents selon qu’on a moins 3 kg, ou entre 3 et 10 kg, ou plus de 10 kg. Evidemment, plus on en prend, moins c’est cher.

Voilà comment cela s’écrit en Java. On commence avec la condition 1 qui est ici et si la condition 1 est vraie, on fait la branche 1 qui est là. Ensuite, si cette condition est fausse, on passe au else et c’est là, ici, qu’on introduit la deuxième condition. Si elle est vraie, c’est la branche 2, mais on ne regardera cette condition que si on n’est pas allé déjà dans la branche 1, donc, si la première condition est fausse. On est dans le else ici de la condition 1 qui est là. Et si la condition 2 est fausse, on va dans le else de la condition 2. Donc, on peut ici relier à chaque condition son else. C’est le else de la dernière condition qui a été introduite.

Donc, est-ce que c’est vraiment ce que je vous ai annoncé, un aiguillage à trois branches ou plutôt une suite de deux aiguillages à deux branches ? Effectivement, c’est plutôt ce deuxième cas qui est mis en œuvre, qu’un aiguillage à trois branches à proprement parler.

Exemple à quatre branches, j’introduis une troisième condition. Pour quatre branches, trois conditions, donc c’est exactement le même principe. J’ai ma première condition avec son else, ma deuxième condition avec son else, ma troisième condition avec son else. Donc, si je prends par exemple cette condition-là, je suis passé dans le else de la première condition, donc je sais que cela c’est faux. Je suis dans le else de la deuxième condition, cela c’est faux. En revanche, je suis dans le cas if de la troisième condition, donc je sais que cela c’est vrai. Donc, j’ai un poids qui n’est pas inférieur à 3 kg, qui n’est pas inférieur à 10 kg mais qui est inférieur à 50 kg. Voilà comment je peux m’y retrouver.

Et à la fin, j’ai un else sans rien. C’est le cas où toutes les conditions sont fausses.

Si je passe maintenant dans l’abstrait, avec des conditions abstraites, si j’ai n branches, j’ai n listes d’instructions : la liste d’instructions 1, la liste d’instructions 2, la liste d’instructions n-1, la liste d’instructions n et j’ai n-1 conditions qui sont ici : la condition 1, la condition 2, la condition n-1. J’ai toujours une condition de moins que de branches parce que le dernier else n’a pas de condition.

Alors sauf si, dans le cas où toutes les conditions sont fausses, je n’ai rien à faire, je peux rejoindre le cas à n branches, je peux ici supprimer cela et j’ai à ce moment-là n-1 branches et n-1 conditions. C’est aussi du Java possible et correct.

Voilà, l’aiguillage à 1 branche peut se représenter comme cela, comme en fait une suite de n-1 aiguillages à deux branches et à chaque fois on a la condition, on a les deux branches : vrai, faux. Et quand c’est vrai, on ne va pas voir les conditions suivantes.

Voilà les différentes variantes qu’on peut avoir sur cette instruction if et qui sont très utiles, qu’on utilise très souvent.

3.5

Bonjour.

Cette vidéo est consacrée à un approfondissement sur les conditions qui sont utiles en Java pour l’instruction if ou pour l’instruction for, et pour aussi d’autres instructions que nous verrons plus tard.

Donc les instructions if et for ont une condition qui est : soit vraie, soit fausse. Selon les cas, les instructions exécutées sont différentes, puisque si la condition est vraie, on fait une suite d’instructions ou l’autre dans un if, ou un tour de boucle de plus ou pas dans un for.

Donc les conditions sont souvent basées sur des opérateurs de comparaison. On a notamment les opérateurs d’égalité et de différence, égal égal et point d’exclamation égal. On a aussi les relations d’ordre, donc plus petit, là, ce devrait être plus grand. Ces opérateurs de comparaison, on peut les utiliser pas seulement pour les nombres, mais également par exemple pour les caractères, pour les booléens, et il y a une exception à cette histoire, c’est que ces opérateurs, on ne peut pas les utiliser avec le type String, le type des messages. Donc par exemple, je ne peux pas savoir si Antoine est plus petit que Alexandre. C’est pourtant une notion qui existe : il existe un ordre sur les chaînes de caractères, mais les opérateurs Java qu’on a là ne permettront pas de savoir si Antoine est avant ou après Alexandre dans l’ordre qu’on appelle l’ordre lexicographique, qui est l’ordre du dictionnaire.

Un exemple d’utilisation de condition avec les caractères: on déclare une variable en réponse, et on demande au vendeur de patates : « est-ce que c’est un bon client ? » On prend la réponse au clavier, on la met dans la variable. Et on teste : est-ce que reponse est égale à O ? Si reponse est égale à O (c’était une réponse en oui/non, donc O signifie oui), on multiplie par… on donne un prix de 2,40 : c’est le prix pour les bons clients. Pour les gens qui ne sont pas de bons clients, ce sera 2 point 6.

On peut avoir des conditions complexes, qui sont basées sur des opérateurs permettant de combiner plusieurs conditions simples. En Java, on a cette écriture : le et logique qui s’applique à deux conditions et s’écrit avec deux et commerciaux. Et le ou logique s’écrit avec deux barres (il faut taper là deux caractères barres qui se trouvent au clavier, dans les gravures spéciales).

Le genre de condition complexe, c’est de dire : il y a plus de 3 kg de patates, et c’est un bon client. Donc en Java, le poids est supérieur à 3 et la réponse à la question quand on lui demande : est-ce que c’est un bon client ?, c’est égal égal au caractère O. Donc une condition complexe comme cela, avec deux conditions et un et entre les deux, cela peut s’utiliser dans un if. Je peux dire : si le poids est supérieur à 3 et que c’est un bon client, je lui donne un tarif encore meilleur. Donc ces opérateurs logiques, quand on a mis un et entre deux conditions, la condition complexe qu’on a là est vraie si les deux conditions, condition 1 et condition 2, sont vraies.

Dans une condition complexe où on fait un ou entre deux conditions simples, il suffit qu’une des deux, ou les deux conditions soient vraies. Donc soit condition 1, soit condition 2, soit les deux conditions à la fois sont vraies.

Une condition correcte, en fait, c’est quoi en Java ? C’est n’importe quel calcul qui renvoie true ou false. Et des types de calculs, on en a plusieurs. On a certes les opérateurs de comparaison, ce sont les conditions les plus fréquentes. Mais cela pourrait être l’appel d’une méthode, par exemple la méthode isDigit, qui nous teste un caractère, et qui nous dit si ce caractère correspond à un chiffre : isDigit nous répondra vrai ici, parce que 1 entre quotes est un caractère qui correspond à un chiffre. Donc cela, c’est une méthode qui renvoie vrai ou faux. Et donc on peut utiliser cela dans un if. On pourrait utiliser une variable : si on avait déclaré une variable boolean varbool, on pourrait utiliser la variable varbool comme calcul pour une condition. Et enfin, une valeur de type boolean comme true ou false, c’est aussi un calcul qu’on peut utiliser dans une condition.

Un mélange de différents éléments, on peut mélanger une variable ici avec un calcul d’opérateur, donc on mélange, si vous voulez cela et cela, dans une condition complexe avec un ou. Prenons un exemple avec une variable. Je déclare une variable rep de type boolean, et je demande : « Entrer un booléen ». MOOC point readBoolean va lire au clavier la réponse, et donc là, ma condition, c’est rep. A la ligne précédente, j’ai lu au clavier soit true, soit false. Si j’ai lu true, rep vaut true et ma condition est vraie. Et dans ce cas-là, cela va afficher c’est true. Si rep contient false, ce sera le cas else qui s’exécutera : c’est false. Et la condition rep tout court, c’est équivalent à tester : est-ce que rep est égal à true ? C’est équivalent. Pourquoi ? Parce que les deux conditions ici, rep et rep égal true ont toujours la même valeur. Si rep vaut true,  rep égal true vaut true. Si rep contient false, rep égal égal true vaut false.

Les conditions sont un élément très important des programmes, et c’est un réservoir de bugs. C’est souvent en écrivant les conditions qu’on se trompe. Donc il faut faire très attention, bien comprendre le et et le ou. Ce n’est pas très compliqué, mais cela demande une attention particulière.

3.6

Je vous propose un exemple de programme avec un if à 3 voire 4 branches. C’est encore une variante de notre programme Patates, dans lequel le marchand diversifie ses activités. Il vend non seulement des patates, mais aussi des carottes et des navets. Et chacun de ces légumes a son prix qui est différent.

Je vais appeler mon programme Légumes, et je vais déclarer mes variables. Je vais avoir comme d’habitude un poids, qui est un nombre à virgule, et pour calculer le prix du sac, il faudra savoir le poids, mais aussi, de quel légume il s’agit. Et là, je vous propose d’utiliser un caractère, char : je vais l’appeler typelegume.

Là, je vais mettre un petit commentaire dans mon programme, c’est une note pour moi (le programmeur) pour, quand je le lirai, me rappeler que C, c’est pour carotte, N navet, et P pour patate.

Ensuite, je peux avoir le résultat prixTotal et je vais faire mes entrées en demandant les deux entrées successivement. readDouble, ensuite, je vais demander « Quel légume ? », et je vais rappeler le code : c ,n ou p.

typelegume MOOC point readChar puisque c’est un caractère qui me donne le type de légume. Et ensuite, je vais faire un if avec une branche pour chacun des types de légume.  If typelegume est égal à p : dans ce cas-là, prixTotal va être égal à poids fois 2.60, on a l’habitude du prix des patates, c’est 2,60. Sinon, si c’est une carotte (ou une botte ou un sac de carottes), il va falloir que j’aie le prix des carottes. Les carottes, je vous les fais à 1,80. Et les navets (typelegumes), les navets ? je ne sais pas… Je vais les faire à 2,20.

Et après, est-ce que je vais distinguer un cas de plus ? Je pourrais distinguer le cas où… Donc si je suis dans le cas else là, je sais que toutes les conditions sont fausses, c’est-à-dire typelegume égal P est faux, typelegume égal C est faux, typelegume égal N est faux. Cela veut dire que la personne n’a entré ni C, ni N, ni P et donc pour moi, c’est une erreur. Et je vais me permettre de dire : « Ce n’est pas un légume connu », voilà.

Maintenant, j’ai un problème qui va se poser, c’est que… Dans ce cas, je suis en train, là de donner le résultat mais, j’ai un souci quand même qui fait que ce programme n’est pas correct. C’est que j’ai calculé le prix total pour les patates, les carottes et les navets, mais s’il y a une erreur et que ce n’est pas un légume connu, je n’ai pas de prixTotal. D’ailleurs, là, il ne va pas être content à la compilation, il va me dire : prixTotal peut ne pas avoir de valeur.  On va essayer de compiler, alors, il faut le sauver : oui. Alors Légumes, je l’enregistre, voilà. Alors, il me dit : ligne 20, il y a un problème. Je ne peux pas afficher prixTotal parce qu’il peut ne pas avoir de valeur. Oui, il peut ne pas avoir de valeur, parce que j’ai mis une valeur ici, mais si je suis dans le cas else, je ne suis pas passé dans cette branche. J’ai mis un prixTotal ici ou là, mais je peux être dans le cas else, auquel cas je n’ai pas donné de valeur à prixTotal. Donc il faut que je fasse quelque chose ici. Peut-être le plus simple… Je vais re-tester…

Je vais mettre ici typelegume égal x. Et là, je vais tester : if typelegume différent de x, alors je sais que je n’ai pas eu cette erreur. Quand il y a une erreur, j’ai mis typelegume à x, juste pour savoir après : est-ce qu’il y a eu une erreur ou pas ? Non.

Si c’est différent de x, je peux faire mon affichage parce que j’ai été dans une des autres branches de if, une des trois premières de ce if à 4 branches. Je compile : il n’est toujours pas content. Pour lui,il y a la même erreur que tout à l’heure, donc il s’imagine que prixTotal peut ne pas être initialisé. Et là, Java se trompe. Mais comme il est têtu, même si je sais que c’est moi qui ai raison, je n’arriverai pas à le convaincre. Donc je n’essaie pas, et je vais juste initialiser ici, à une valeur quelconque. Je sauve, je compile, et cette fois, cela marche.

Je vais exécuter mon programme pour tester qu’il marche bien. Je mets un poids de 1 kilo pour des carottes. Je m’attends à ce que ce soit 1,80 ? Eh oui, c’est 1,80. Je ré-exécute avec un poids de 1 point 0 : des navets, je m’attends à ce que ce soit 2,20. Vous voyez, mon programme marche.

Donc on est dans un exemple de if à 4 branches. Je ne suis pas très satisfait de ce que j’ai écrit ici. C’est une façon de contourner le programme, qui marche. Je ne suis pas très satisfait parce que j’ai utilisé typelegume pour autre chose que ce qu’il doit contenir. J’utilise typelegume comme un témoignage qu’il y a une erreur. Et je ne trouve pas cela bien. Typelegume, cela doit être le type de légume. Si j’ai besoin de gérer une erreur, je préfère utiliser une autre variable, et je choisis une variable boolean : ok, que j’initialise à true. Et donc là, au lieu d’utiliser typelegume avec un code bidon, je vais mettre : ok égal false. Et là, ma condition, cela va être ok tout court. Ou alors, si vous voulez, si vous insistez beaucoup, vous pouvez mettre ok égal égal true. Mais ok égal égal true, cela a la même valeur que ok tout court. Ok c’est true ou false. Si ok vaut true, ok égal égal true vaut true, si ok vaut false, ok égal égal true vaut false. Donc on peut écrire cela tout simplement, et je re-sauve mon programme, je le recompile, et je pense qu’il va fonctionner aussi bien. Alors, pour 1 kilo de navets, il va faire prixTotal. Et si je prends 1 kilo de K : ce n’est pas légume connu (pas très joli), « ce n’est pas un légume connu » : voilà.

Voilà donc qui termine avec cet exemple de if à 4 branches.

3.7

Donc tu vois-là, je suis entrain de faire une petite réussite sur un simulateur de téléphone, et tout cela est en java comme la plupart des applications qui tournent sur les téléphones.

François BARTHELEMY : est-ce que tu peux nous montrer les applications sur ce téléphone pour qu’on voie ce que c’est ?

Olivier PONS : oui, si on revient là on a toutes les applications, navigateurs, et tout ça, tout cela est écrit en java, enfin la principale, le texte-là comme tout ce que vous avez sur les tablettes c’est écrit en java, enfin tablettes androïdes entendons-nous bien, l’autre grand système donc qui est AOIS par Apple est plutôt écrit dans un autre langage qui est Objectif-C.

François BARTHELEMY : et tu insistes bien sur le fait que ce sont les applications qui sont écrites en java, cela veut dire que le système lui-même n’est pas écrit en java.

Olivier PONS : oui, le système androïde n’est pas écrit en java, ce sont des choses beaucoup plus compliquées, mais tout ce qui est interface pour programmer, des jeux, des navigateurs ou tout ce qu’on voudra se fait en java.

François BARTHELEMY : je vais te présenter rapidement pour nos amis qui ne te connaissent pas encore, tu es Olivier PONS, tu enseignes au CNAM la programmation des téléphones, tablettes et autres…

Olivier PONS : voilà, tout ce qu’on appelle programmation mobile, donc ça va des téléphones aux montres connectées et tablettes.

François BARTHELEMY : d’accord, tu sais que là on tourne une vidéo pour un MOOC java et est-ce que le java de ces applications c’est le java normal, le java qu’on a appris dans ce MOOC ?

Olivier PONS : alors globalement oui, c’est un java tout à fait normal, d’ailleurs on verra que les passes des compilations se font normalement, maintenant il y a des librairies spécifiques donc des morceaux de programmes qu’on va charger qui sont spécifiques. Mais globalement, c’est du java normal avec des choses en plus.

François BARTHELEMY : et d’après ce que tu m’as expliqué c’est du java normal quand on écrit. Mais après ce qu’on en fait au cours de la compilation c’est un peu différent.

Olivier PONS : oui donc le code java tel qu’on l’écrit est transformé, d’habitude en java c’est transformé vers certains types de langages, là il y a une étape après qui transforme en quelque chose qui va être spécifique à Androïd, je ne rentrerais pas dans les détails, globalement cette chose-là est ensuite installée dans le téléphone qui va la faire tourner.

François BARTHELEMY : donc si je reprends par rapport à ce que nos amis connaissent, si on fait le parallèle avec DRjava, on écrit comme dans DRjava dans l’éditeur et on compile comme on compile dans DRjava. Là ça nous donne un fichier compilé, mais l’histoire en DRjava s’arrête là alors qu’en Androïd c’est encore compilé une deuxième fois.

Olivier PONS : c’est exactement cela

François BARTHELEMY : et après c’est encore compilé une troisième fois sur le téléphone lui-même.

Olivier PONS : Alors sur le téléphone ce n’est plus compilé, c’est ce qu’on appelle interprété, c’est-à-dire qu’il va faire marcher la version compilée qu’on a installée. Tout à fait exacte, dans les versions récentes, c’est encore plus rapide et y produire du code natif c’est-à-dire du code pour le téléphone en question.

François BARTHELEMY : Et donc ça impose qu’il y ait une phase de compilation sur le téléphone.

Olivier PONS : oui

François BARTHELEMY : Et donc pour nous convaincre que c’est bien du java, tu as écrit une petite application, qui est la petite application du MOOC, tu peux nous la montrer sur le téléphone ?

Olivier PONS : oui, alors sur le téléphone je peux vous la montrer, voilà, elle est là, c’est une petite icône, si je clique dessus, on a une petite application et on va taper un ongle, et je vais mettre 4, je lance ça m’écrit quatre fois du beau code. C’est vraiment pour vous montrer comment ça marche. Une chose importante, voyez c’est graphique par rapport à ce que vous avez déjà vu c’est-à-dire qu’ici on a vraiment un écran graphique avec des images, on appelle ça une vue. Et que plutôt que de lancer un programme avec une fonction "main" comme vous avez l’habitude de le faire, ici on a des programmes qui réagissent à des événements. C’est quoi un événement, c’est le fait qu’un utilisateur clique, c’est l’utilisateur qui intervient, mais ça peut aussi être le système, quelque chose a fini de ce charger. Il est dix heures si vous programmez votre réveil, des choses comme ça.

Voilà donc, cette petite application on peut regarder le code qui lui correspond. Donc on n’est pas dans DRjava, on est dans un autre environnement pour écrire du code qui s’appelle" Androïd studio" qui ressemble mais qui est plus compliqué que DRjava parce qu’il y a beaucoup de choses à écrire, en fait il y a toute cette présentation, les boutons, les choses comme ça qu’on peut écrire d’un côté, et puis il y a ensuite du code java comme vous avez l’habitude de l’écrire. Donc si on prend ce petit exemple, ici vous avez un IF, comme vous avez l’habitude de l’écrire on fait une comparaison entre un nombre et la valeur 1 et si ce nombre est plus grand que la valeur 1, on va faire l’action qui est ici qui est d’ajouter un "s".

François BARTHELEMY : si je peux reprendre un peu, vous voyez comme vous avez l’habitude des variables entières, des variables "string" le "IF" que vous connaissez, le" For" que vous connaissez aussi donc, vous voyez bien que c’est le genre de codes que vous avez l’habitude d’écrire, je crois qu’on ne rentrera pas plus dans les détails pour cette petite application. Si ce n’est que la boucle" for" vous l’avez compris sert à afficher autant de fois du beau code que le nombre qu’on a rentré à la source.

Olivier PONS : si on revient à notre petite démonstration, ici dans le simulateur, on a si on met 6, nous aurons bien 6x6, c’est la boucle "for" qui se charge de faire cet affichage alors qui est ici minimaliste.

François BARTHELEMY : et comme cette application est très intéressante, on vous proposera de la télécharger sur le, vous aurez,

Olivier PONS : de l’installer sur votre téléphone, pour faire joujou , voilà, ce qui est aussi intéressant de remarquer c’est que si on a un simulateur, donc ceci n’est pas spécifique à Androïd c’est dès qu’on fait de la programmation mobile pour pouvoir tester les différents appareils existants plutôt que de brancher directement une tablette ou un téléphone.

François BARTHELEMY : on fait le programme sur l’ordinateur avec un degré de confort supérieur, et une fois qu’il est fini on le transfère sur la tablette ou sur le téléphone. Bien, je crois que c’était intéressant et donc je te remercie d’être venu nous parler de ça.

Olivier PONS : voilà, et puis si vous voulez en savoir plus là dessus, il y a des formations donc vous l’avez compris à la programmation mobile au CNAM, donc je vous encourage à aller regarder ce qui s’y fait, voilà.

4.1

Nous allons voir maintenant la boucle while, qui est une deuxième boucle Java, qui est plus simple que la boucle for. Comme la boucle for, elle est basée sur une condition. Et le principe de la boucle while, c’est d’exécuter une suite d’instructions tant que cette condition est vraie.

Voilà comment cela s’écrit : avec le mot-clé while, la condition entre parenthèses comme dans un if, est entre accolades, la suite d’instructions qu’on veut exécuter plusieurs fois. L’exécution du while commence par le calcul de la condition. Si la condition est fausse, la boucle est terminée. Vous allez me dire : On vient juste de la commencer, oui. Mais si la condition est fausse, la boucle est terminée quand même. Si la condition est vraie, on exécute la suite d’instructions une première fois. Et ensuite, on recalcule la condition. Si la condition est fausse, c’est terminé. Si la condition est vraie, on exécute une deuxième fois la suite d’instructions, etc.

Voilà comment on peut schématiser l’exécution d’une telle instruction : on commence avec le début de programme, avec des instructions l’une après l’autre, on arrive là. La condition, si elle est vraie, on va de ce côté. Donc, là encore, c’est un aiguillage. C’est comme un if, un aiguillage avec une voie quand c’est vrai, une voie quand la condition est fausse. Mais la différence avec un if, c’est qu’après avoir exécuté la suite d’instructions, on revient à la condition et on la recalcule à nouveau. Alors que le if, lui, il nous emmènerait tout de suite à la suite du programme. Donc, chaque fois qu’on arrive à cette condition : elle est vraie ou fausse. Imaginez : elle est vraie ; je fais un premier tour ; je reviens là ; elle est vraie : deuxième tour, je viens là, elle est vraie : troisième tour, je viens là, elle est fausse, j’ai terminé, j’ai fait 3 tours de boucle, j’ai calculé 4 fois la condition. Les 3 premières fois, elle était vraie, mais la 4ème fois, elle était fausse.

Je vais prendre comme exemple la vente de patates, une seule exécution pour tous les clients de la matinée. Je prends le début du programme Patate. Alors mon programme, je l’appelle PatateMatin, je lui donne un nom différent et je reprends le début qui est inchangé. En revanche, je vais faire une nouvelle entrée dans mon programme Patate, c’est que, après avoir calculé le prix d’un sac, je demanderai au marchand s’il a terminé sa matinée, ou s’il continue avec un client suivant. Donc, je vais utiliser pour cette communication, un caractère. Je vais l’appeler fin et cela me dira si le travail est fini et cela pourra prendre le caractère N si ce n’est pas fini ou le caractère O, si c’est fini. Et je vais continuer à calculer des prix de patates tant que fin est égal à N. Cette condition-là, je peux la reformuler en disant : tant que je n’ai pas fini, je fais le travail habituel qui consiste à demander le poids, lire le poids, afficher le prix. Là, j’ai choisi d’afficher directement le calcul sans le mettre dans une variable intermédiaire. Ensuite, il me reste comme travail à savoir si, une fois ce prix calculé, c’était le dernier, on plie le matériel dans le camion, ou si on continue. Donc, je demande à l’utilisateur : A-t-il fini ? et je lui précise que je veux qu’il me réponde O  ou N. Et dans la variable fin, je mets ce que la personne a tapé avec la méthode  readChar. Je ferme la boucle while. Donc c’est cela qui va être exécuté en boucle jusqu’à ce que la personne ait tapé oui pour dire : j’ai fini. Donc, quand dans la variable fin il y a O, fin égal égal N : la condition va être fausse.

Deuxième exemple. Pour ce deuxième exemple, on va prendre une nouvelle méthode qui permet de tirer au sort un nombre entier. MOOC point auSort : c’est une méthode qui va tirer un nombre entier entre 1 et 6 dans cet exemple (on lui donne en paramètre le plus petit et le plus grand entier possibles dans le tirage au sort). Donc, quand je tire un nombre entre 1 et 6, là, cela va avoir le même effet que de tirer un dé. Voici donc mon exemple avec un while. Je prends un nombre. Ici, j’ai une déclaration suivie d’une affectation. Donc je déclare la variable nombre et je tire un nombre au sort entre 1 et 10, cette fois. Ce n’est pas un dé, c’est entre 1 et 10. Et tant que mon nombre est différent de 10, je retire au sort. Je refais ce qui est fait ici : le tirage au sort et mettre le résultat dans nombre, je le refais à chaque tour de boucle.

Voyons un exemple d’exécution de ce programme. Je vais noter ici ce qui est tiré au sort, la mémoire et les instructions exécutées.

La première instruction exécutée, cela va être la déclaration. La déclaration ici, a pour effet de me réserver un espace mémoire pour nombre. Et le tirage au sort, là, je suppose que c’est 9 qui est tiré au sort. Donc, 9 est mis dans nombre, on le retrouve dans la mémoire.

Deuxième instruction : la condition : nombre différent de 10. Est-ce que 9 est différent de 10 ? Oui. Donc la condition est vraie, je fais un tour de boucle de plus.

Nombre égal MOOC point auSort, je suppose que cette fois, c’est 7 qui est tiré au sort.

4ème instruction : on revient à la condition voir si elle est vraie. Est-ce que 7 est différent de 10 ? Oui, la condition est vraie, donc un tour de plus. Le tour de boucle de plus, c’est un tirage au sort, on suppose que cette fois, c’est 10. 6ème instruction : est-ce que nombre est différent de 10 ? Est-ce que 10 est différent de 10 ? Non, la condition est fausse, la boucle est terminée et le programme s’arrête.

Quelles sont les différences entre while et for? Déjà, cela se ressemble beaucoup. C’est à peu près interchangeable. Par exemple, si j’ai un for, comme cela, avec une initialisation, une condition, une incrémentation, un corps, je peux le réécrire en un while équivalent. L’initialisation que j’ai là, je la fais avant de rentrer dans la boucle. La condition ? C’est la même. Je recopie le corps et l’incrémentation, je la mets à la fin de la boucle. Si je fais cela, j’ai du Java strictement équivalent. Dans l’autre sens, si j’ai un while, je peux écrire un for un peu étrange. C’est un for qui peut vous étonner, parce qu’ici, je n’ai pas d’initialisation, ici, je n’ai pas d’incrémentation. C’est néanmoins du Java correct. Donc, je peux réécrire un while dans un for équivalent.

Mais ce que je vous propose, c’est d’avoir une discipline pour utiliser l’un ou l’autre selon le cas. Et la règle que je vous propose, c’est : au moment de commencer la boucle, est-ce qu’on connaît le nombre de tours à faire ? Je précise bien que c’est au moment de l’exécution de la boucle. Ce n’est pas au moment où j’écris le programme. Si au moment d’exécuter la boucle, de commencer son exécution, je connais le nombre de tours à faire, il faut utiliser un for. Et si on ne connaît pas le nombre de tours à faire, utiliser un while.

Voilà. Donc nous allons, à partir de maintenant, utiliser l’une ou l’autre boucle, selon nos applications.

4.3

Nous allons voir ce qui peut mal se passer avec les boucles lors de l'exécution. Il y a deux cas de figure possibles : soit des boucles qui se terminent beaucoup trop vite, soit des boucles qui ne se terminent jamais.

Les deux boucles qu'on connaît, le for et le while,sont contrôlées par une condition. Et si la condition est vraie, on fait un tour de boucle de plus, qui consiste à exécuter une séquence d'instructions. Si la condition est fausse, la boucle est terminée. Il y a deux cas particuliers avec ces conditions, le cas d'une condition qui est toujours fausse, et le cas d'une condition qui est toujours vraie.

Prenons le premier cas d'une condition toujours fausse. Si la condition est fausse dès le début, il n'y a même pas de premier tour de boucle. Prenons un exemple avec une boucle for. Ici dans cette boucle for, on initialise le numéro de tour à 1. Et la condition pour continuer, c'est que num doit être négatif ( num inférieur à 0 cela peut se redire : num est négatif). Oui, mais si num est initialisé à 1, il n'est pas négatif. Donc la boucle s'arrête tout de suite, avant même d'avoir fait le premier tour. Num est initialisé à 1, et quand on se demande: est-ce que num est inférieur à 0 ? Non, 1 n'est pas inférieur à 0 : la condition est fausse, la boucle ne s'exécute pas. Un autre exemple, cette fois avec un while. On initialise une variable rep de type caractère avec N. Et on se dit : tant que rep est égal à O. Évidemment, quand on exécute cette première ligne, cela nous met N dans rep et ensuite, on se pose la question : est-ce que la condition est vraie ? Est-ce que rep est égal à O ? Non, c’est égal à N, donc la condition est fausse et là encore, rien n'est exécuté par la boucle. Donc aucun tour de boucle n'est exécuté. Dans ce cas-là, forcément, la boucle ne sert à rien. C'est un bug. Parce qu'on ne va pas commencer à écrire des instructions qui ne servent à rien.

Autre cas, qui est encore plus embêtant, c'est une condition qui est toujours vraie. On fait toujours un tour de boucle en plus, le programme ne s'arrête jamais. Voyons un exemple. Une boucle for avec num qui est initialisé à 1, et on continue tant que num est positif, cette fois. La condition, c'est : num est positif. Voyez, au début il est égal à 1, ensuite, on rajoute 1. Donc il devient de plus en plus grand. Un nombre positif, quand on lui rajoute 1, il devient plus grand et il n'a aucune raison de devenir négatif. Donc, cette boucle ne s'arrête jamais, num devient de plus en plus grand, et à chaque fois, on écrit num à l'écran. Autre exemple, cette fois avec une boucle while. On initialise num à 1, et on continue tant que num est inférieur à 10. Donc num vaut 1 au début, et après, num ne change plus jamais. Vous voyez qu'on n'a pas d'instruction ici num égal, num égal : un certain calcul. Donc num ne change plus jamais et, à chaque tour de boucle, qu'est-ce qu'on fait ? On affiche le contenu de num : 1. Cela va nous afficher 1, puis 1, puis 1 : cela ne s'arrête jamais. La condition reste toujours vraie. Donc, une condition toujours vraie, généralement, c'est un bug. Les conditions, pour être intéressantes, doivent être parfois vraies, parfois fausses.

Il y a quand même des exceptions de programmes qui ne doivent pas s'arrêter, des programmes qui doivent tourner éternellement. Des exemples de ces programmes, c’est par exemple des programmes qui contrôlent des feux tricolores pour la circulation. On veut qu'ils marchent nuit et jour, 24 heures sur 24. Autre exemple, le contrôle d'une centrale nucléaire : on ne veut pas que le contrôle s'arrête, pour quelque raison que ce soit. Les systèmes d'alarme, c'est pareil : on veut que le programme tourne en permanence. Dans ces cas-là, on peut utiliser un while, mais on mettra comme condition while(true) : true, c'est une condition qui est toujours vraie. C'est un calcul booléen, et le fait d'écrire while(true), cela veut dire que, quand on le lira, on saura que cette boucle-là, on ne souhaite pas qu'elle se termine.

On peut avoir des cas un peu plus subtils. C'est que la condition peut être vraie ou fausse, mais quand elle est vraie, après elle ne change plus. Donc, une fois qu'elle devient vraie, elle ne change plus et cela veut dire que le programme ne se termine pas non plus. Voici un exemple. Il y a un nombre entier qui est rentré au clavier, et on fait une boucle tant que ce nombre est supérieur à 10. Donc là, on ne sait pas si le nombre est supérieur ou inférieur à 10, cela dépend de ce que l'utilisateur va taper au clavier. S'il rentre quelque chose de plus grand que 10, on va faire la boucle, et on reviendra à la condition qui sera toujours la même chose. Si par exemple, c'est 12, la deuxième fois qu'on réévalue la condition, c'est toujours 12, la troisième fois, c'est toujours 12 : donc la condition est toujours vraie. En revanche, si l'utilisateur tape 5, la condition est fausse. Donc on est dans un cas d'une condition qui peut être soit vraie, soit fausse, mais une fois vraie, elle ne change plus.

Donc cela, c'est un indice de bug. La condition doit finir par devenir fausse pour chaque exécution du programme. C'est-à-dire, qu'on veut que le programme se termine quoi qu'il arrive, pour toute exécution. Pour cela, il faut que le corps de la boucle et/ou l'incrémentation change un élément de la condition.

Cela étant, cela arrive de se tromper, par exemple, oublier une incrémentation dans une boucle while ; oublier de faire un calcul qui remet les choses à jour, et donc on peut faire des programmes qui bouclent. Et si, lors d'une exécution, on teste le programme, le programme ne se termine pas, il faut l'arrêter de force. Là, il y a plusieurs solutions. Pour interrompre l'exécution, par exemple, en drjava, on a un bouton reset qui permet d'arrêter. Et quand on est dans un terminal, on peut faire un control-c, ou alors on a une version plus violente qui consiste à fermer la fenêtre dans laquelle s'exécute le programme. Ce sont des choses qu'on peut être amené à faire.

Donc, conclusion, quand on écrit une boucle, il faut faire un certain nombre de vérifications : vérifier que la condition peut être vraie la première fois, et vérifier qu'un tour de boucle change quelque chose dans la condition. Si ces deux conditions sont vraies, c'est qu'on a des boucles bien faites, et des programmes qui se comporteront comme on le souhaite, c'est-à-dire qui s'exécuteront. Après avoir exécuté des instructions, ils s'arrêteront finalement.

4.4

Nous allons voir quelques notions pour concevoir les programmes, en sachant que concevoir et réaliser un programme, c'est une vaste question qui mériterait peut-être un MOOC à elle toute seule. La difficulté est proportionnelle au problème traité et ici, on va s'attacher à des petits problèmes, avec des hypothèses de départ qui sont qu'on a un programmeur seul, un problème simple à énoncer et à résoudre, et donc là, on est vraiment dans de l'artisanat, dans de l'apprentissage.

Je vous propose donc de procéder en différentes étapes. Première étape: énoncer le but. Deuxième étape : identifier les entrées et les sorties du programme. Etudier les interactions avec l'utilisateur. Quatrième étape : identifier les autres données du programme. Cinquième étape : identifier les résultats intermédiaires à stocker en mémoire. Etape 6 - vous voyez, cela arrive assez tard - : écrire les déclarations de variables. Donc, c'est juste à partir de cette étape 6 qu'on commence à faire du Java. Etape 7: écrire les autres instructions. Etape 8 : compiler. Et étape 9, tester : quelque chose que je vous invite à ne pas négliger.

On va reprendre donc les étapes une par une.

Première étape : énoncer le but. S'il s'agit d'un exercice, comme par exemple ceux qui vous sont proposés dans ce MOOC, énoncer le but, c'est déjà fait : c'est l'énoncé de l'exercice. Et si ce n'est pas le cas, ce que je vous propose, c'est d'écrire un énoncé, comme si c'était un exercice, pour exprimer ce que vous voulez faire.

Deuxième étape : identifier les entrées et les sorties. Donc, pour nous, les entrées viennent du clavier, les sorties vont à l'écran. Chaque entrée doit avoir un type Java. Par exemple, le poids d'un sac de patates est, par nature, un nombre à virgule. Et pour ce qui est des sorties, c'est sous quelle forme on veut afficher les sorties, quels genres d'interactions, de message on veut pour rendre les choses éventuellement plus jolies, plus claires.

Etudier les interactions avec l'utilisateur : cela vient du fait qu'on peut avoir différentes interactions pour les mêmes entrées et les mêmes sorties. Je vais prendre comme exemple : calculer le prix de plusieurs sacs de patates. On a des entrées qui sont le poids de n sacs de patates, et si on a n sacs de patates, on aura des sorties pour les prix, il y aura n prix différents. Avec ce schéma de n entrées/n sorties, on a plusieurs schémas d'interactions possibles, et je peux les schématiser au moyen de petits graphiques.

Dans ces graphiques, je mets l'utilisateur d'un côté, le programme de l'autre, et donc, le premier type d'interaction possible, c'est : le programme demande le premier poids, l'utilisateur donne le premier poids, et le programme répond en affichant le prix du premier sac. Ensuite, le programme demande le poids du deuxième sac, l'utilisateur le rentre, etc., jusqu'à avoir fait cela n fois. Autres types d'interactions possibles : le programme demande le premier poids, l'utilisateur rentre le premier poids. Le programme demande le deuxième poids, l'utilisateur rentre le deuxième poids, etc., jusqu'à ce que la personne ait rentré le nième poids. Et là, le programme affiche les différents prix.

Alors vous allez me dire que ce n'est pas ce comportement qui est souhaité. Non, ici, certainement, ce n'est pas le bon schéma d'interaction, mais avec les mêmes entrées et les mêmes sorties, on pourrait faire un programme qui se comporte comme cela. Pour décrire ces interactions, je vous propose d'utiliser un diagramme de séquence, donc qui détaille les interactions pour une exécution ou pour un morceau d'exécution, avec ces deux acteurs qui sont l'utilisateur, le programme, et les choses qui vont dans un sens ou dans l'autre. Dans un sens, c'est un affichage, dans l'autre, c'est une saisie au clavier. Eventuellement, si le programme a différents comportements selon certaines conditions, on peut faire autant de schémas que de besoin pour étudier les différents types de comportements.

L'étape suivante : identifier les autres données. Donc, les entrées, elles viennent du clavier, mais on peut avoir besoin d'autres données, qui sont écrites en dur dans le programme, comme par exemple, le prix du kilo dans le programme Patate. Donc ces données, c’est important d'identifier ces données, elles sont nécessaires pour le calcul et c'est important de les recenser.

Les résultats intermédiaires : donc, des résultats de calculs où on a commencé à utiliser des données pour avoir des calculs, et on peut vouloir les stocker dans des variables s'ils sont utilisés plusieurs fois dans la suite du programme. On a un résultat intermédiaire, on veut l'utiliser plusieurs fois. Plutôt que de le recalculer à chaque fois qu'on en a besoin, on le met dans une variable. Ou alors, on peut mettre un résultat intermédiaire dans une variable pour le mettre en valeur, pour faciliter la lecture du programme. Donc dans ces deux cas-là, on va utiliser une variable.

L'étape suivante : une fois qu'on a recensé donc les entrées, les autres données et les résultats intermédiaires qu'on souhaite stocker, on va déclarer les variables. Ce que je vous conseille, c'est de mettre toutes les déclarations au début du programme, avec une exception : les numéros de tour d'une boucle for, qui sont déclarés dans la boucle for. Une variable pour chaque entrée, pour chaque résultat intermédiaire, et optionnel : variables pour les sorties et variables aussi pour les autres données du programme.

Ensuite, on commence à écrire les instructions. Et ce qui structure le programme, ce sont les boucles et les if. Donc, commencer par écrire les boucles et les if. Et parmi les choses importantes, donner des noms explicites aux variables, bien indenter les instructions, c'est-à-dire décaler les suites d'instructions qui sont dans les if, dans les boucles. Et toujours privilégier la lisibilité. La lisibilité, c’est quelque chose d'important. Cela facilite à la fois la mise au point et la maintenance du programme et c'est vraiment quelque chose d'important. C'est plus important d'avoir un programme lisible que d'avoir un programme efficace. La lisibilité, c’est vraiment quelque chose qu'il faut travailler en permanence, avoir en permanence présent à l'esprit. L'efficacité, en revanche, c'est quelque chose qui concerne certaines applications, et souvent même, certains morceaux d'une application. Il y a des endroits où c’est important d'être efficace, et il y a beaucoup d'autres endroits où cela n'a pas d'importance réelle ; il n'y a pas d'enjeu réel, parce que, qu'on fasse un programme ou un autre, il n'y aura pas de différence visible.

Alors ensuite, une autre question : ces différentes étapes, est-ce qu'on les fait sur papier ou sur ordinateur ? Alors, là, j'ai une proposition simple, c'est : ce qui se fait plus facilement sur papier, on le fait sur papier. Par exemple, les dessins. Les cas d'utilisation, c'est plus facile de les gribouiller sur un bout de papier. Le code Java, manifestement, il faut le faire sur l'ordinateur, parce que, il faudra le faire tôt ou tard : autant le faire directement sur ordinateur, à moins que vous ayez vraiment une prédilection pour le papier, vous préférez tout écrire avant de commencer à taper. Le texte en français, là, vous avez le choix, selon l'outil que vous préférez. Par exemple, pour l'énoncé, vous pouvez le taper dans votre traitement de texte préféré, ou le mettre sur un bout de papier.

Plus tard, évidemment, toutes les entrées ne viendront pas du clavier. Il y aura aussi les souris, les boutons, les menus, les différentes choses d'interface graphique. Il y aura les fichiers, les bases de données, les réseaux, les autres programmes, les capteurs. Toutes les sorties n'iront pas à l'écran : il y aura les fichiers, les bases de données, les réseaux, les imprimantes et les autres programmes. Voilà.

C'est donc quelques conseils assez schématiques pour vous guider dans la conception de vos petits programmes.

Nous allons voir quelques notions pour concevoir les programmes, en sachant que concevoir et réaliser un programme, c'est une vaste question qui mériterait peut-être un MOOC à elle toute seule. La difficulté est proportionnelle au problème traité et ici, on va s'attacher à des petits problèmes, avec des hypothèses de départ qui sont qu'on a un programmeur seul, un problème simple à énoncer et à résoudre, et donc là, on est vraiment dans de l'artisanat, dans de l'apprentissage.

Je vous propose donc de procéder en différentes étapes. Première étape: énoncer le but. Deuxième étape : identifier les entrées et les sorties du programme. Etudier les interactions avec l'utilisateur. Quatrième étape : identifier les autres données du programme. Cinquième étape : identifier les résultats intermédiaires à stocker en mémoire. Etape 6 - vous voyez, cela arrive assez tard - : écrire les déclarations de variables. Donc, c'est juste à partir de cette étape 6 qu'on commence à faire du Java. Etape 7: écrire les autres instructions. Etape 8 : compiler. Et étape 9, tester : quelque chose que je vous invite à ne pas négliger.

On va reprendre donc les étapes une par une.

Première étape : énoncer le but. S'il s'agit d'un exercice, comme par exemple ceux qui vous sont proposés dans ce MOOC, énoncer le but, c'est déjà fait : c'est l'énoncé de l'exercice. Et si ce n'est pas le cas, ce que je vous propose, c'est d'écrire un énoncé, comme si c'était un exercice, pour exprimer ce que vous voulez faire.

Deuxième étape : identifier les entrées et les sorties. Donc, pour nous, les entrées viennent du clavier, les sorties vont à l'écran. Chaque entrée doit avoir un type Java. Par exemple, le poids d'un sac de patates est, par nature, un nombre à virgule. Et pour ce qui est des sorties, c'est sous quelle forme on veut afficher les sorties, quels genres d'interactions, de message on veut pour rendre les choses éventuellement plus jolies, plus claires.

Etudier les interactions avec l'utilisateur : cela vient du fait qu'on peut avoir différentes interactions pour les mêmes entrées et les mêmes sorties. Je vais prendre comme exemple : calculer le prix de plusieurs sacs de patates. On a des entrées qui sont le poids de n sacs de patates, et si on a n sacs de patates, on aura des sorties pour les prix, il y aura n prix différents. Avec ce schéma de n entrées/n sorties, on a plusieurs schémas d'interactions possibles, et je peux les schématiser au moyen de petits graphiques.

Dans ces graphiques, je mets l'utilisateur d'un côté, le programme de l'autre, et donc, le premier type d'interaction possible, c'est : le programme demande le premier poids, l'utilisateur donne le premier poids, et le programme répond en affichant le prix du premier sac. Ensuite, le programme demande le poids du deuxième sac, l'utilisateur le rentre, etc., jusqu'à avoir fait cela n fois. Autres types d'interactions possibles : le programme demande le premier poids, l'utilisateur rentre le premier poids. Le programme demande le deuxième poids, l'utilisateur rentre le deuxième poids, etc., jusqu'à ce que la personne ait rentré le nième poids. Et là, le programme affiche les différents prix.

Alors vous allez me dire que ce n'est pas ce comportement qui est souhaité. Non, ici, certainement, ce n'est pas le bon schéma d'interaction, mais avec les mêmes entrées et les mêmes sorties, on pourrait faire un programme qui se comporte comme cela. Pour décrire ces interactions, je vous propose d'utiliser un diagramme de séquence, donc qui détaille les interactions pour une exécution ou pour un morceau d'exécution, avec ces deux acteurs qui sont l'utilisateur, le programme, et les choses qui vont dans un sens ou dans l'autre. Dans un sens, c'est un affichage, dans l'autre, c'est une saisie au clavier. Eventuellement, si le programme a différents comportements selon certaines conditions, on peut faire autant de schémas que de besoin pour étudier les différents types de comportements.

L'étape suivante : identifier les autres données. Donc, les entrées, elles viennent du clavier, mais on peut avoir besoin d'autres données, qui sont écrites en dur dans le programme, comme par exemple, le prix du kilo dans le programme Patate. Donc ces données, c’est important d'identifier ces données, elles sont nécessaires pour le calcul et c'est important de les recenser.

Les résultats intermédiaires : donc, des résultats de calculs où on a commencé à utiliser des données pour avoir des calculs, et on peut vouloir les stocker dans des variables s'ils sont utilisés plusieurs fois dans la suite du programme. On a un résultat intermédiaire, on veut l'utiliser plusieurs fois. Plutôt que de le recalculer à chaque fois qu'on en a besoin, on le met dans une variable. Ou alors, on peut mettre un résultat intermédiaire dans une variable pour le mettre en valeur, pour faciliter la lecture du programme. Donc dans ces deux cas-là, on va utiliser une variable.

L'étape suivante : une fois qu'on a recensé donc les entrées, les autres données et les résultats intermédiaires qu'on souhaite stocker, on va déclarer les variables. Ce que je vous conseille, c'est de mettre toutes les déclarations au début du programme, avec une exception : les numéros de tour d'une boucle for, qui sont déclarés dans la boucle for. Une variable pour chaque entrée, pour chaque résultat intermédiaire, et optionnel : variables pour les sorties et variables aussi pour les autres données du programme.

Ensuite, on commence à écrire les instructions. Et ce qui structure le programme, ce sont les boucles et les if. Donc, commencer par écrire les boucles et les if. Et parmi les choses importantes, donner des noms explicites aux variables, bien indenter les instructions, c'est-à-dire décaler les suites d'instructions qui sont dans les if, dans les boucles. Et toujours privilégier la lisibilité. La lisibilité, c’est quelque chose d'important. Cela facilite à la fois la mise au point et la maintenance du programme et c'est vraiment quelque chose d'important. C'est plus important d'avoir un programme lisible que d'avoir un programme efficace. La lisibilité, c’est vraiment quelque chose qu'il faut travailler en permanence, avoir en permanence présent à l'esprit. L'efficacité, en revanche, c'est quelque chose qui concerne certaines applications, et souvent même, certains morceaux d'une application. Il y a des endroits où c’est important d'être efficace, et il y a beaucoup d'autres endroits où cela n'a pas d'importance réelle ; il n'y a pas d'enjeu réel, parce que, qu'on fasse un programme ou un autre, il n'y aura pas de différence visible.

Alors ensuite, une autre question : ces différentes étapes, est-ce qu'on les fait sur papier ou sur ordinateur ? Alors, là, j'ai une proposition simple, c'est : ce qui se fait plus facilement sur papier, on le fait sur papier. Par exemple, les dessins. Les cas d'utilisation, c'est plus facile de les gribouiller sur un bout de papier. Le code Java, manifestement, il faut le faire sur l'ordinateur, parce que, il faudra le faire tôt ou tard : autant le faire directement sur ordinateur, à moins que vous ayez vraiment une prédilection pour le papier, vous préférez tout écrire avant de commencer à taper. Le texte en français, là, vous avez le choix, selon l'outil que vous préférez. Par exemple, pour l'énoncé, vous pouvez le taper dans votre traitement de texte préféré, ou le mettre sur un bout de papier.

Plus tard, évidemment, toutes les entrées ne viendront pas du clavier. Il y aura aussi les souris, les boutons, les menus, les différentes choses d'interface graphique. Il y aura les fichiers, les bases de données, les réseaux, les autres programmes, les capteurs. Toutes les sorties n'iront pas à l'écran : il y aura les fichiers, les bases de données, les réseaux, les imprimantes et les autres programmes. Voilà.

C'est donc quelques conseils assez schématiques pour vous guider dans la conception de vos petits programmes.

4.7

Je vous propose un exemple de programme avec une boucle while. Ce programme, c'est un petit jeu où l'ordinateur va tirer un chiffre au sort entre 1 et 10 (un nombre, d'ailleurs, si c'est 10, c'est un nombre.). Et l'utilisateur va chercher à le deviner. Et là, typiquement, on ne sait pas à l'avance combien de tours de boucle il faudra faire. Au pire, il faudra faire 10 tours de boucle, redemander à l'utilisateur une nouvelle tentative s'il n'a pas deviné au bout de la 10ème fois. Mais on ne sait pas, peut-être qu’il trouver au premier coup. Donc un while plutôt qu'un for.

Donc là, j'ai mis le squelette d'un programme. Je vais lui donner comme nom, Deviner et je vais commencer. Cela va être un nombre entier, donc type int. Je vais appeler cela nombre, et je vais utiliser MOOC point auSort pour tirer au sort un nombre entre 1 et 10. Et puis, je vais avoir besoin d'une autre variable pour contenir l'essai de l'utilisateur, donc je vais l'appeler essai. Et puis, je vais faire une boucle tant que essai est différent de nombre, on va demander à l'utilisateur de rentrer son essai. Essai égal MOOC point auSort de 1,10. Alors, cette boucle n’est pas mal. Il y a quand même un problème, c'est que la première fois, je vais essayer de comparer essai à nombre, alors que essai n'a pas de valeur. Je vais l'initialiser pour que la condition soit fausse au départ. Nombre a été tiré au sort entre 1 et 10, si je veux être sûr que essai soit différent de nombre, je peux mettre par exemple moins 1 comme initialisation. Et à la fin, je peux mettre un petit message en disant : « bravo, c'était bien ça » Je vais sauver mon programme Deviner, OK, je le compile. Cela a marché. Je vais l'exécuter. Oh, il y a comme un problème ! Alors, c'est quoi ? Ah oui ! Là, il ne faut pas que je tire au sort le résultat, il faut que je le demande à l'utilisateur. Bien sûr, bien sûr, bien sûr. MOOC point readInt, ce sera beaucoup mieux si je lui demande de le prendre au clavier plutôt que de le tirer au sort. Alors, je ne sais pas du tout : 5, ce n'est pas cela. 1 ? Voilà. J'ai trouvé au deuxième essai. Je vais recommencer. Là, de toute façon, c'est effectivement tiré au sort. Donc je n'ai pas à tricher sur la démo, je ne sais pas quel nombre doit être trouvé. Je recommence une deuxième fois : 5, 7, 8, 9, 1, 2, 3, 4 ? J'ai tout essayé, non ? 6. Voyez, là, c'est vraiment le cas où on n'a pas de chance, j'ai dû tout essayer pour trouver le nombre. Voilà.

Maintenant, je vous propose de faire une variante de ce programme, où on l'améliore. Je vais l'appeler DevinerPlus, où on va demander à l'utilisateur quel nombre max il souhaite. Je vais avoir besoin d'une nouvelle variable. Cela, je préfèrerais le mettre là. Je déclare mes variables en début de programme, et je lui demande quel max il souhaite : max égal MOOC point readInt. Donc, je vais utiliser max à la place de 10 dans mon tirage au sort. Ici, je déclare seulement la variable et je ferai le tirage au sort ici : nombre égal auSort de 1 max. Donc cette fois, ce ne sera plus entre 1 et 10, mais entre 1 et max. Voilà. – on va essayer - J'aurais voulu le sauver dans un autre fichier : save as DevinerPlus. Je le compile, je l'exécute. Alors cette fois, je vais prendre 3, par exemple : un nombre entre 1 et 3. Et voyez, il me demande un nombre entre 1 et 3 : « Bravo, c'était bien ça. » Et là, c'est une façon de tricher si je demande un nombre max entre 1 et 1. Il n'y en a pas beaucoup. Je tape 1, et il me dit : « Bravo, c'était bien ça ». Donc, ce n'est pas très fair-play de ma part, mais, je l'ai fait. Trop tard. Voilà.

Donc, voilà un exemple de programme qui utilise un while.

5.1

Bonjour. Je vais vous présenter, dans cette vidéo, l'intérêt des tableaux, le tableau qui est le gros sujet de cette semaine.

Ce qu'on ne peut pas encore faire avec ce qu'on connaît déjà, c'est appliquer le même calcul à plusieurs données du même type. Ce qu'on voudrait, c'est pouvoir faire le même calcul, éventuellement un calcul compliqué avec toute une suite d'instructions, sans avoir à le réécrire plusieurs fois si c'est la même chose. On voudrait l'écrire une seule fois. Mais actuellement, une donnée, elle est dans une variable, et une variable a un nom, et ce nom intervient dans le calcul. Donc par exemple, si mon but c’est d'augmenter de 10 % le prix de tous les légumes, je vais avoir une variable pour le prix de chaque légume : une pour le prix tomates, une pour le prix patates, etc., et faire mon augmentation de 10 %. A chaque fois, c'est un calcul différent, parce que j'ai le nom, ici, qui est différent : prixPatate, prixTomates fois 1 virgule 1. Ce qu'on voudrait, c'est pouvoir faire, pour tous les prix de légumes, augmenter le prix de 10 %. Donc, on voudrait faire par exemple, une boucle for. Mais on a ce problème du nom de variable, qui est différent à chaque tour.

On a un autre problème aussi, avec nos données, c'est que le nombre de données varie d'une exécution à l'autre. Par exemple… L'exemple dont je vais vous parler, c'est le ticket de caisse. Un ticket de caisse, c'est un certain nombre d'articles, et chaque article a son prix. Tant qu'il s'agit de rajouter des prix, on n'a pas besoin de mémoriser. Mais si on veut par exemple faire une opération qui est l'opération d’annuler, si le client dit : « Ah mais non, celui-là, finalement, je ne le veux pas », on ne peut pas annuler un article si on ne l'a pas mémorisé avant. Donc, il faut garder cela dans des variables, si on n'a que les variables. Ces variables ont des noms différents, et on risque de ne pas avoir suffisamment de noms selon le nombre d'articles qu'a pris le client.

Donc la solution à ces problèmes, c'est de passer par un système de numérotation des données qui ont une même nature, et la possibilité de faire une boucle pour appliquer un calcul à toutes ces données, possibilité de pouvoir écrire un calcul en prenant en compte la numérotation des données.

Donc le programme de la semaine, c'est cette grande nouveauté que sont les tableaux. Et puis, on va approfondir sur les entrées/sorties, c'est-à-dire, ce qu'on affiche à l'écran/ce qu'on prend au clavier. Et on va aussi voir les boucles imbriquées, c'est-à-dire les boucles dans les boucles. Et un peu de savoir-faire aussi : comment faire un menu.

Voilà le programme que je vous propose pour cette semaine et donc, bon travail.

5.2

Bien. Nous allons voir dans cette vidéo comment créer et utiliser des tableaux, les tableaux qui sont une structure permettant de regrouper plusieurs données du même type.

Prenons un exemple : là, c'est un tableau qui contient des nombres entiers. Donc, ce qui est important, c'est que dans un tableau comme cela, on ne peut mettre que des choses qui sont du même type au sens Java. Donc là ici, ce sont des int, et le tableau est dans la mémoire de l'ordinateur. Il est stocké dans la mémoire de l'ordinateur, et les cases sont numérotées. La première case du tableau est numérotée à 0, et la dernière case, cela dépend de la taille du tableau. On peut avoir des tableaux plus ou moins grands. Là, j'ai 5 cases. Mon premier numéro, c'est 0, et le dernier, c'est 4.

On peut faire des choses en considérant le tableau entier. En fait, ce ne sont pas vraiment des opérations qu'on peut faire, on peut faire essentiellement des affectations. On peut aussi utiliser une case du tableau. Et une case du tableau, son contenu, c'est comme une variable. C'est un nombre ici, donc c'est un tableau qui contient des nombres, donc le contenu d'une case, c'est un nombre.

Le tableau a une vie qui comprend 3 étapes. La première étape, c'est la déclaration d'une variable qui donne un nom au tableau. Donc c'est une déclaration de variable normale, c'est-à-dire que tab, c'est le nom que j'ai choisi, et ici, c'est son type. Et le type, là : int (alors cela ne se voit pas forcément très clairement, mais c'est int crochet ouvrant, crochet fermant : c'est cela qui est écrit ici), cela désigne le type des tableaux contenant dans chaque case un int.

L'étape 2, c'est de créer le tableau dans la mémoire et de l'affecter à la variable qu'on a définie précédemment. Donc, new int, c'est une instruction qui a pour effet de créer le tableau dans la mémoire. A partir de ce moment-là, le tableau existe, et le égal, c'est l'affectation qu'on connaît déjà, qui permet d'associer une valeur à un nom. Donc, au nom tab, on va associer le tableau entier. 5 qu'on trouve ici, c'est le nombre de cases et quand on crée le tableau, toutes les cases contiennent 0.

Cela, c'était donc la deuxième étape, créer le tableau. Une fois que le tableau est créé, on peut l'utiliser pour y stocker des choses. Alors, comment s'utilise un tableau? Chaque case du tableau peut être utilisée comme une variable du même type. Et le nom d'une case, on l'a en mettant le nom du tableau plus le numéro entre crochets. Donc tab de 1, c'est le nom de la case numéro 1, et ce nom-là : tab crochet 1 est utilisable comme si c'était une variable de type int. La case numéro 1, c'est la deuxième case du tableau, parce que la première case est le numéro 0. Ce nom de case, je vous le dis, cela peut s'utiliser comme une variable, cela veut dire que cela peut apparaître à gauche d'une affectation. Donc, si j'écris tab de 1 égal 14, cela aura pour effet de mettre 14 dans la 2ème case du tableau. Et la case, on peut l'utiliser comme une variable aussi dans un calcul. Par exemple, ici, je peux la passer en paramètre à une méthode. Je pourrais aussi l'utiliser dans un calcul arithmétique, par exemple.

Prenons un exemple où on voit donc les différents temps de la création et de l'utilisation du tableau, les 3 étapes.

Etape 1: on définit une variable, tab donc cela la crée en mémoire.

2ème étape : on crée le tableau, le new int de 3 crée le tableau. Trois, c'est le nombre de cases, donc ce tableau est en mémoire avec les numéros 0, 1, 2, pour les différentes cases, et l'affectation, qui l'associe au nom tab. Dans la mémoire, tab va recevoir ce tableau.

Ensuite : affectation. Dans la case qui a pour numéro 1, on met la valeur 5. Donc la valeur 5 se retrouve ici.

Dernière instruction, MOOC point readInt, on va lire quelque chose au clavier, et le mettre dans la case numéro 2. Au clavier, on lit 3, c'est cela qui se retrouve dans la case n° 2.

Le numéro de case, tel qu'il apparaît dans un programme, peut être le résultat d'un calcul. Voici un exemple où le numéro de case n'est pas donné avec le nombre directement, mais avec un calcul. On commence, comme tout à l'heure, à déclarer le tableau. Deuxième ligne, le tableau est créé. Ensuite, on déclare une variable de type int, qui s'appelle num, num pour numéro. Dans la case n° 1 du tableau, on met 5, dans la case n° 2, on met ce qui est lu au clavier, puis dans la case, (on va afficher une case du tableau), et on demande à la personne, à l'utilisateur, quelle case il veut afficher. On le lit au clavier, on le met dans la variable num. Et ensuite, on va chercher la case numéro num dans le tableau tab. C'est ici. Et le contenu de cette case est affiché par system out point println. Donc là, vous allez me dire que ce calcul est simple. J'aurais pu aussi écrire tab de num plus 1. Cela aurait été un autre calcul, et cela aurait été également du Java correct, où le numéro de case, donc, n'est pas donné directement par le numéro mais par un calcul qui permet d'obtenir un nombre entier qui sert de numéro de case.

On utilise souvent for pour manipuler les tableaux, et le numéro de tour de boucle correspond au numéro de case du tableau. Et dans ce cas-là, si on veut parcourir tout le tableau, il faut commencer à numéroter à partir de 0. Donc, il faut commencer à numéroter les tours de boucle de la boucle for à partir de 0. Et le dernier tour de la boucle for doit avoir le numéro de la dernière case. Ce numéro, le numéro de la dernière case, c'est la taille du tableau moins un. Ici, il y a 5 cases, le dernier numéro, c'est 4. Donc, cela, c'est quelque chose d'assez général.

Un exemple qui illustre comment on fait un for, donc là, la tâche, c'est remplir un tableau avec des nombres.

Comme tout à l'heure, on crée un tableau. C'est-à-dire, on déclare la variable et on crée le tableau en une seule instruction, et on part sur la boucle for.

Donc on commence à numéroter à 0, pour avoir le numéro de la 1ère case, et on s'arrête avant d'arriver à 5. Donc, pour continuer, il faut que le num soit strictement inférieur à 5. Donc c'est assez stéréotypé, c'est souvent comme cela qu'on fait. Ici, on met 0, c'est la 1ère case. Et ici, ce qu'on met, c'est la taille du tableau, mais avec un inférieur strict. Une autre solution équivalente, ce serait de faire inférieur ou égal à 4. Ce serait une autre condition équivalente. Je reviens. Et donc, à chaque tour de boucle, on demande d'entrer un nombre, et on utilise ici le numéro de tour de boucle. Donc au début, au 1er tour de boucle, mon num vaut 0. Je vais remplir la case tab de 0. Ensuite, au 2ème tour de boucle, j'aurai incrémenté de 1. Donc, num vaudra 1. Et donc, au 2ème tour de boucle, c'est tab de 1 que je vais remplir avec quelque chose que je lis au clavier, etc., jusqu'à avoir mes 5 nombres.

Pour chaque type Java, on peut faire un tableau. Donc on peut avoir les tableaux de booléen, les tableaux de char, les tableaux de double, les tableaux de string, qui s'écrivent comme ça, avec des crochets derrière. Donc cela, ce sont les noms des types. Je peux utiliser par exemple, ce nom de type et puis donner le nom d'une variable. Donc, quand on parle d'un tableau de string, les strings, c'est ce qu'il y a dans les cases. Mais cela ne change pas les numéros : les numéros de cases restent des nombres. Les numéros de case sont des int. Et à la création du tableau, il y a une valeur bien spéciale dans chaque case. Si c'est un tableau d'entiers, il y a 0 dans chaque case. Si c'est un double, c'est 0 point 0 dans chaque case, si c'est boolean, c'est false dans chaque case. Et si c'est char et string, ce sont des choses bizarres dont je n'ai pas envie de vous parler, parce qu'on est des débutants, donc je ne veux pas parler de choses compliquées.

Les opérations : sur une case, on met les opérations du type du contenu. Donc par exemple, sur un tableau d'entiers, j'ai les opérations du type int ; les opérations arithmétiques ; les comparaisons d'ordres ; je peux utiliser la case comme paramètre pour une méthode.

Sur un tableau complet, il n'existe pas d'opération à proprement parler. Ce n'est pas un opérateur comme un opérateur arithmétique qui permet de faire un calcul. En revanche, on peut faire une affectation sur un tableau entier. Donc, voyez ici je déclare un premier tableau et je crée un tableau de 3 cases. Je déclare un deuxième tableau et ils ont le même type, ce sont deux tableaux d'entiers. J'ai créé donc un tableau tab1, et dans tab1 de 0, je vais mettre 10. Et ensuite, je fais l'affectation. Dans tab2, je mets le tableau que j'appelle tab1. Et donc, cette affectation, c'est le tableau entier : tab1 c'est le tableau qui est créé là, tab2, c'est aussi ce tableau-là. Et quand je fais System out point println de tab2 de 0, cela va m'afficher quoi ? Cela va m'afficher 10. C'est dans tab1 que j'ai mis 10, mais après, j'ai transféré le tableau de tab1 dans tab2, donc tab2 de 0, c'est la même chose en fait que tab1 de 0. Donc on peut faire une affectation d'un tableau entier.

La taille d'un tableau, c'est le nombre de cases, et c'est fixé à la création du tableau, quand on fait un new. Quand je fais un new, je mets entre crochets new int de 150. Et cela, c'est la taille. Et une fois le tableau créé, la taille ne peut pas changer. En revanche, la taille n'est pas fixée lors de la déclaration de variables, donc je peux changer : je peux mettre des tableaux de tailles différentes dans une variable à différents moments du programme.

Un exemple : je prends une variable tab. Je lui affecte un tableau de 5 cases. Le new int, cela me fait un tableau à 5 cases, comme ça, où dans chaque case, j’ai 0. Étape suivante : je mets 10 dans la 1ère case. Donc ici, à la place de 0, je mets 10. Ensuite, je crée un nouveau tableau de 10 cases. Donc, cette fois, il y a 10 cases, je ne dessine pas tout, mais il y a 0 partout. Et j'affecte cela donc, à la variable tab. Cela veut dire que tab, au début, c'était cela, (avant cette affectation, c'était cela) mais depuis que j'ai créé mon nouveau tableau, tab c'est devenu cela. Et quand j'affiche tab de 0, cela m'affiche 0. Le 10 que j'avais mis dans ce tableau-là, il est toujours dans ce tableau-là, mais cela, cela ne s'appelle plus tab de 0.

Donc, ma variable n'a pas de taille en tant que telle. En revanche, un tableau qui est créé, il a une taille et cette taille ne peut pas changer.

Voilà qui termine cette introduction sur les tableaux, que nous allons maintenant beaucoup utiliser.

5.3

Cette vidéo est consacrée aux entrées et sorties, donc ce qui se passe au clavier et à l’écran, et c’est en partie des révisions, mais on va quand même voir des choses nouvelles.

Donc, les entrées se font au clavier avec MOOC qui a une librairie spécifique à ce cours. Les sorties se font à l’écran en faisant un system out point print qui est du java standard.

La seule différence, c’est parce qu’il n’y a pas de java standard qui fasse les entrées au clavier de façon pratique et simple.

Les sorties à l’écran, elles existent pour les différents types : int, double, char, boolean et string, avec deux formes, la forme avec printt et la forme avec println: println passe à la ligne, print ne passe pas à la ligne.

Et donc tous les types qu’on connaît peuvent être affichés sauf les types tableaux. Il n’y a pas de system out point print pour les tableaux et si on veut afficher un tableau, il n’y a pas d’autre solution que de faire une boucle for et afficher les cases l’une après l’autre avec l’affichage : si c’est un tableau d’int avec system out point println; si c’est un tableau de double ou un tableau de char, pareil : system out point println ou print si on veut mettre toutes les cases sur la même ligne.

En n’utilisant pas le ln, on peut afficher plusieurs choses sur la même ligne et, grâce à ce genre de choses, on peut sauter une ligne blanche. Alors, là, ce qu’il y a, ici, c’est les doubles quotes directement l’une à côté de l’autre, sans rien entre les deux. Donc, c’est le message avec aucun caractère dedans. Mais on peut aussi éventuellement mettre un message avec un caractère blanc mais à l’affichage, ce sera le même, même si on met deux, trois, cinq, dix caractères blancs. Afficher du blanc, cela aura le même effet et du fait qu’on passe à la ligne, cela sautera une ligne blanche.

Alors, prenons un exemple de plusieurs affichages sur la même ligne. Là on a le prixTotal, c’est une variable, on lui donne un certain nombre. On commence à faire afficher et là voyez, on affiche sans ln, sans ln, et on ne saute à la ligne qu’à la fin. Donc, on va commencer par afficher prixTotal, ensuite, le contenu de la variable et ensuite euros. Donc, c’est cela qui s’affiche à l’écran.

Les entrées au clavier, c’est comme les sorties. Elles existent pour les types simples, les types de base. Pour chaque type, on a une méthode MOOC : readXxxx. Xxxx, c’est à remplacer par ( Int, Double, Char, Boolean ), avec la majuscule après le read.

Donc, ces méthodes là en entrée, les méthodes de MOOC ne lisent qu’une seule valeur par ligne. On ne peut pas rentrer plusieurs chiffres sur la même ligne, et taper  return cela ne marchera pas. Et elles peuvent provoquer une erreur si ce qui est rentré au clavier n’est pas du bon type. Donc par exemple, si c’est readBoolean, et que je tape un nombre, cela provoquera une erreur. Et comme pour les affichages, il n’y a pas de méthode pour lire un tableau entier, il faut lire case par case pour remplir le tableau petit à petit.

Un exemple scolaire, lire un tableau de noms, lecture et écriture du tableau. Donc on commence par créer le tableau en le déclarant un tableau de string et on fait un tableau de quatre cases. Ensuite, on fait des boucles for pour parcourir ce tableau, on commence à 0, on va jusqu’à quatre et on parcourt toutes les cases. Quand je dis on va jusqu’à 4, 4 exclu, on s’arrête à 4. Il y a 4 cases et le dernier numéro c’est 3. Et pour chaque numéro de case qui est un numéro de tour de boucle, on affiche entrer le nom sans ln, ensuite le numéro. Donc, le nom 0, le nom 1, le nom 2, le nom 3 : c’est cela qui s’affichera aux différents tours et on va lire donc au clavier là, il y aura une lecture et on remplira la case numéro num du tableau. Donc, ce tableau va être rempli case par case.

La deuxième boucle for, c’est pour l’afficher et là, on a décidé de l’afficher sur la même ligne. Donc, c’est la même boucle for en ce qui concerne l’entête. Vous retrouvez ici le même entête que là et ce qui change, c’est qu’au lieu de remplir le tableau, on va lire, afficher le tableau. Donc, on commence par afficher le contenu de la case num et ensuite on met un espace parce que si on ne met pas d’espace, les noms seront collés les uns aux autres à l’écran, donc on met, là j’ai dû mettre deux ou trois caractères espace dans ce message pour séparer, vu que ce sont des print tout courts et pas des println. Tout le tableau, les quatre noms du tableau seront affichés sur la même ligne.

Une petite démo. Je vais, dans ma démo, surtout insister sur les erreurs qui peuvent se produire. Pardon…

Donc, l’entête de programme que vous connaissez bien, sans comprendre, et je déclare par exemple un int X et je vais le lire au clavier avec un mooc point readInt. Je vais le sauver - erreurIO - , c’est parfait. Je le compile. Cela a marché. Je l’exécute. Il attend que je tape un nombre. Vous voyez, d’habitude, dans les programmes, j’avais soin de donner un message pour dire ce que j’attendais, ici je n’ai pas fait d’affichage. Dans ce programme-là, il n’y a pas d’affichage à l’écran, donc, il faut que je sache, moi, qu’il attend que je rentre un entier. Je rentre un entier, 12 par exemple. Le programme s’arrête, cela marche. Je voulais vous montrer ce qui se passe et je tape autre chose qu'un entier. Je tape a : le programme s’arrête et il y a un message d’erreur. Donc, cela c’est une erreur à l’exécution : input mismatch exception. On n’a pas besoin forcément de tout comprendre. Il nous dit des choses là qu’on ne peut pas comprendre, mais il y a un endroit où on peut avoir une information intéressante,

c’est ici : erreurIO point java, cette ligne-là. Pourquoi ? Parce que erreurIO point java, c’est ce qu’on a écrit au-dessus et il nous dit que c’est ligne 4 le problème. Cela doit être cela la ligne 4 : - 1, 2, 3, 4 -.

Donc, c’est pareil pour les autres types. Si on attend par exemple un boolean qui entre un chiffre, c’est une erreur de ce genre-là qui apparaîtra à l’écran.

On va continuer maintenant avec un dispositif intéressant pour faire des affichages, c’est qu’on a la concaténation de chaînes qui est utilisable de façon assez pratique. Un petit rappel, la concaténation de chaînes qui s’écrit plus permet de concaténer deux chaînes de caractères, donc bon plus jour, cela va nous donner la chaîne bonjour. Et ce qu’il y a de sympathique avec cette concaténation, c’est qu’elle marche aussi entre un string et une valeur d’un autre type. Donc, si je concatène un string et un int, cela va me donner un string qui contiendra cet entier. Donc, par exemple, nombre plus 1, nombre ici est un string,  1 c’est un int, cela va me donner exactement ce que je peux désirer, la chaîne nombre1. Pareil, là j’ai choisi de mettre la chaîne après et avant un booléen, cela va marcher, cela va me faire falselune. Et là, si je fais deux concaténations à la suite ou je concatène le Prix total, un nombre et une autre chaîne, cela me donne une chaîne qui contient le résultat entier que je veux afficher. Donc, cela c’est une façon d’écrire assez rapidement les résultats qu’on souhaite.

Donc, pour notre exemple Patate, on peut remplacer les deux lignes où on affiche d’abord un message, ensuite un nombre par une seule ligne et on concatène les deux ici.

Voilà ce que je voulais vous dire à propos des entrées et sorties.

5.4

Dans cette vidéo, nous allons voir comment faire un menu pour gérer les interactions entre le programme et l'utilisateur. Dans les applications modernes, les interactions se font via une interface graphique avec souris, bouton menu, champs de texte, mais nous, on n'a pas tout cela. On a juste afficher et lire du texte, de façon séquentielle, c'est-à-dire qu'on peut juste mettre le texte à la suite, éventuellement passer à la ligne. C'est tout ce qu'on peut faire. On ne peut jamais remonter, on ne peut jamais se balader pour revenir intercaler quelque chose. Donc, avec ces contraintes, l'outil de base pour communiquer avec l'utilisateur, c'est le menu texte.

On va partir sur l’exemple du vendeur de patates qui diversifie son activité. Donc on va proposer un menu de ce genre-là, où il y aura un sac et on va proposer trois choix : sac de patates, sac de carottes ou sac de poireaux, et l'utilisateur devra entrer 1, 2 ou 3 pour dire quelle option il a choisi. Les prix sont différents, donc en fonction du choix, le calcul va être différent. Et donc, il faut adapter le programme Patate.

Donc, l'affichage du menu en Java, c’est juste des System out point println. Là, on ne met pas de ln, mais c'est un peu du détail, vous mettez un passage à la ligne si vous voulez ici. Et on a besoin donc d'une variable pour connaître le choix de l'utilisateur. Le choix de l'utilisateur ici, on a décidé de lui faire taper un nombre, donc on a choisi un int.

La prise en compte du choix, après: c'est un if, avec autant de branches qu'il y a de choix dans le menu. Si le choix est égal à 1, on met ici le prix (alors je ne me rappelle plus, c'était quoi?) les patates, les carottes et les poireaux. Donc, ici c'est le prix des patates, ici le prix des carottes, ici le prix des poireaux, et donc cela correspond : les patates, les carottes, les poireaux. Après, on affiche donc le prix.

Maintenant, que se passe-t-il si la personne n'a tapé ni 1, ni 2, ni 3 ? Dans ce cas-là, on n'a pris aucune branche du if. Et dans ce cas-là, le prixTotal n'a pas été calculé, et on ne peut pas l'afficher. Donc, quand on fait un menu, souvent, il faut vérifier aussi que le choix entré est un choix valide. Donc là, on fait cette vérification et on a une condition là, qui est que si le choix est plus petit que 1, ou plus grand que 3, donc si ce n'est pas dans l'intervalle des nombres qui sont autorisés pour répondre à la question, on affiche « choix incorrect » et sinon, on affiche le prix total. Il y a une autre méthode pour vérifier le choix, on vérifie le choix et on ne lâche pas la personne tant qu'elle n'a pas répondu à la question. Donc la version précédente, c'est : si son choix est incorrect, on lui dit : « Choix incorrect » et on ne fait pas le travail. On ne lui affiche pas de prix, on ne fait pas de calcul. Cette méthode-ci consiste à dire : si le choix est incorrect, donc là, c'est la même condition, et c'est même « tant que le choix est incorrect », on lui dit : « Choix incorrect » et on lui demande d'en rentrer un autre. Et cette boucle, on la fait éternellement, jusqu'à ce qu'il ait répondu : 1, 2, ou 3. On ne le lâche pas.

Maintenant, ce programme avec le menu, on veut le faire en boucle. Donc, le faire en boucle, c'est : afficher le menu ; saisir la réponse ; réaliser le calcul ; afficher le résultat du calcul. Qu'est-ce qu'on va mettre là comme while ? Quelle condition ? Et aussi, quand s'arrêter ? Donc, là, la réponse c'est de proposer de quitter le programme comme une nouvelle option du menu, option 4 : quitter le programme.

Donc le choix : on l’initialise à 0 pour être sûr que la première fois, on va faire l'affichage du menu, il faut que la condition soit vraie la première fois qu'on l'examine quand on commence la boucle. La condition : tant que le choix est différent de 4 et au menu, on rajoute donc cette option 4, pour quitter.

Ce qui nous donne maintenant un programme avec cette structure : on commence donc avec l'initialisation et la déclaration du choix. Auparavant, il faut quand même aussi déclarer la variable poids et la variable prixTotal (c'est à faire avant). La condition, c'est : est-ce que le choix est différent de 4? La première fois, oui. La première fois, on est sûr d'arriver là. Afficher le menu : ce sont des System out point print et des System out point println. Saisir le choix. Ensuite, selon la valeur du choix, si on a un choix qui n'est pas correct, on revient donc, là c'est la boucle qui nous fait la saisie du choix. Et ensuite, on a le if qui nous dit : si on a 1, on fait le calcul pour les patates, si on a 2 : le calcul pour les carottes, si on a 4, c'est qu'on a demandé à quitter et dans ce cas-là, on ne fait pas de calcul, et on revient directement à la boucle ici. Et là, quand notre réponse est 4, choix est devenu égal à 4, on est sûr après d'aller dans cette branche-là.

Donc, ce programme-là : on a une petite boucle pour vérifier que le choix est correct ; une grande boucle pour peser plusieurs sacs de patates, de carottes ; et à chaque tour de boucle, on peut emprunter un des 3 chemins, cela peut être différent à chaque fois, ou cela peut aussi être le même.

Et pour conclure, on va faire la démo de ce programme. Donc on va revoir un peu tout cela, puisqu'il y a beaucoup de choses et je vous ai montré, beaucoup de bouts séparément, on va revoir tout cela un peu à la file.

Donc, on commence par déclarer nos deux variables du programme Patate habituel. La variable choix pour le choix dans le menu : on l’initialise à 0 pour être sûr de faire l'affichage du menu au moins la 1ère fois. Pour que la 1ère fois, le choix, soit différent de 4, il faut qu'il ait une valeur avant de commencer la boucle. On affiche le menu, on lit le résultat.

Ensuite, le choix qui a été lu, s'il est plus petit que 1 ou plus grand que 4, c'est une erreur donc on met un message d'erreur et on lui demande de recommencer. Là, on peut passer à la ligne, ce sera plus joli.

Donc, quand on sort cette boucle while, on est sûr qu'on a 1,2,3 ou 4. S'il y a 4, on ne fait rien, mais si c'est différent de 4, on va faire le calcul. Donc là, on a besoin de ce « if » pour s'assurer que ce ne soit pas le choix de quitter. Si le choix c'est de quitter, on ne va rien faire du tout.

Donc on va demander le poids. Cela, il faut toujours le faire bien sûr, et cela, quel que soit le légume. En revanche, après, le calcul va dépendre du légume. Donc c'est là qu'on va reprendre encore en compte le choix pour voir si c'est patates à 2,60, carottes à 1,70 ou poireaux à 2,10.

Donc on a fait 3 calculs différents, on affiche le prix, et on termine la boucle while, et à la fin, si on veut, après la boucle while, donc quelque part ici, (je vais chercher la fin de la boucle while, c'est là, on peut dire « au revoir », c'est poli.

Je sauve mon programme, je le compile, et il n'est pas content. Pourquoi il n'est pas content ? Parce qu'il se dit : prixTotal n'a pas forcément été initialisé. PrixTotal, je l'ai mis à jour ici et je sais que, quand je suis là, je sais que je suis parti dans une de ces 3 branches j'en suis sûr pourquoi ? Parce que j'ai testé ici que c'était bien entre 1 et 4, et j'ai testé ici que ce n'était pas 4. Donc c'est forcément 1,2 ou 3. Moi, je sais que le prixTotal, il a été initialisé. Mais le compilateur ne le sait pas. Donc, pour cette raison, je suis obligé, ici, de mettre une valeur, n'importe laquelle, pour que le compilateur accepte de me compiler le programme. Si je ne le fais pas, le programme ne se compilera pas, donc on ne va pas se battre. On va lui donner une valeur inutile, et on recompile. Et cette fois, cela a marché.

Je vais agrandir la fenêtre pour l'exécution, puisque cette fois, on a pas mal de choses à afficher, et on y va pour l'exécution. Je vais prendre des patates 1 point 0. Je me trompe, je prends le choix 5. Il me dit : « Choix incorrect » et il faut que je re-choisisse des poireaux : 1 kilo, 2,10. Et je choisis de quitter. Voilà, tout s'est bien passé.

Donc, voilà le genre d'interaction qu'on aura avec l'utilisateur. C'est ce qu'on peut faire de mieux pour le moment, et cela nous permet déjà de faire plein de choses, donc on ne va pas s'en priver.

5.5

Bonjour. Nous allons voir dans cette vidéo qu'on peut mettre des boucles à l'intérieur d'autres boucles, c'est ce qu'on appelle les boucles imbriquées.

Dans le corps d'une boucle for ou d'une boucle while, on a une suite d'instructions, et cette suite d'instructions, cela peut être des instructions simples, comme une affectation ou un affichage. Mais cela peut être une instruction complexe, qui contient elle-même d'autres suites d'instructions, comme un if, un for, ou un while. Voici un petit dessin qui montre un if dans une boucle for. Donc on a la boucle for, avec son initialisation, sa condition, son incrémentation, et le corps de la boucle. Le corps de la boucle, c'est donc une suite d'instructions, et dans cette suite d'instructions, on retrouve la condition, et les 2 branches d'un if. Donc, cela ne pose pas de problème.

2ème exemple : un for dans une boucle for. Donc, on a 2 boucles, qui ont chacune leur initialisation, donc l'initialisation 1 ici pour la boucle 1, et l'initialisation 2 pour la boucle 2. La condition 1 pour la première boucle, la condition 2 pour la boucle qui est à l'intérieur de l'autre, l'incrémentation 1 ici, l'incrémentation 2 là. Et le corps de la boucle 1, c'est tout ce qu'il y a là, donc y compris la boucle for, et le corps de la deuxième boucle est là.

Donc, j'ai bien gribouillé partout, on ne voit plus rien, mais je vais effacer pour montrer donc : que se passe-t-il ? Supposons que la condition est vraie, on va donc de ce côté de la branche. Et si on a une autre condition, supposons qu'elle soit vraie, on arrive là, et c'est la petite boucle, celle qui est à l'intérieur, qui est faite. Elle peut être faite un certain nombre de fois, puis la condition devient fausse, on revient à la condition de la 1ère boucle, celle qui contient l'autre. Et là, si la condition est vraie, on est parti sur un second tour de boucle de la grande. Et puis un 1er tour de la nouvelle petite, etc. Donc on a comme cela, un enchaînement de boucles l'une dans l'autre, et c'est quelque chose qui est assez utile.

Donc, le premier exemple que je vous propose, c'est un compte à rebours, où on veut afficher donc les nombres dans l'ordre inverse : on veut faire 10, 9, 8, 7, 6, etc. Et à chaque tour de boucle, on veut effacer l'écran pour effacer le nombre précédent, et effacer l'écran comme on l’a vu précédemment, c'est déjà une boucle, qui consiste à afficher un certain nombre de lignes blanches. On affiche le nouveau nombre, avant celui qui était précédemment affiché. Faire une petite pause : pour faire une petite pause dans le programme, on introduit une nouvelle instruction, une méthode qui est : MOOC point pause, où on donne entre parenthèses un nombre de secondes pendant lequel il faut patienter. Un nombre de secondes qui peut être un nombre à virgule si on veut moins d'une seconde, par exemple.

Donc voilà comment va s'écrire ce programme : donc, on décide de compter. Et là, ce que dit la 1ère boucle, c'est qu'on va compter avec un premier numéro de tour qui va aller de 1 jusqu'à 10. Donc, on va compter de 1 à 10 les numéros de tour. On commence par effacer ce qui pouvait être sur l'écran et pour cela, on a num2 qui est donc un autre numéro. Chaque boucle a son propre numéro de tour : on a num pour la 1ère boucle, num2 pour l'autre boucle, et là, on va de 1 à 50. De 1 à 50, qu'est-ce qu'on va faire à chaque tour de boucle ? On va afficher une ligne blanche donc afficher 50 lignes blanches, cela efface l'écran.

Donc cette boucle complète qui est à l'intérieur de l'autre, cela commence par effacer l'écran. Ensuite on affiche num et on fait la petite pause : une pause de 4 secondes, c'est un peu long, j'aurais peut-être dû mettre moins. Et cela va nous afficher (mais ce n'est pas ce que je vous avais annoncé, je vous avais annoncé un compte à rebours). Là, si j'affiche num, cela va afficher la première fois 1, la seconde fois 2, 3, 4, donc cela ne sera pas un compte à rebours, ce sera un compte dans l'ordre croissant. Si je veux le compte à rebours comme je l'avais annoncé, il faut rectifier cette erreur : ce n'est pas num ici. Donc, la 1ère fois, cela vaut 1 et je veux que ce soit 10 qui s'affiche. Donc, je vais faire 11 moins num, et la 1ère fois, cela va afficher 10. Au 2ème tour de boucle, num vaudra 2. J'afficherai 11 moins num, 2. Cela fera 9 donc ce qu’il faut que je mette ici à la place de num, c'est 11 moins num. Voilà.

Le 2ème exemple : l'affichage d'un rectangle. Donc je prends comme thème (avec 3 lignes et 4 colonnes), je fais un rectangle comme on fait quand on n'a pas ce qu'il faut pour dessiner, on met un caractère, là j'ai choisi de mettre le caractère étoile pour colorer l'intérieur de mon rectangle. Le principe là, c'est qu'on peut voir ce dessin entre guillemets, comme comportant des lignes et des colonnes. Donc là, j'ai numéroté les 3 lignes, les 4 colonnes. On va faire une boucle pour les lignes, et c'est une boucle qu'on va faire 3 fois, puisqu'il y a 3 lignes, ici. Au 1er tour, on va afficher cela, au 2ème tour, on va afficher cela, au 3ème tour, on va afficher cela. Et on peut faire une autre boucle pour les colonnes. C'est-à-dire, je suis dans la boucle qui doit afficher le 1er tour. Je vais faire une première boucle, une boucle dont le 1er tour va afficher la 1ère étoile, le 2ème tour la 2ème étoile, etc. Donc afficher une ligne, c'est afficher toutes les colonnes de la ligne, et passer ensuite à la ligne suivante.

Je vais vous montrer comment on peut faire cela. Public class Rectangle… Alors, il faut que je fasse une première boucle : je vais mettre nlig par exemple pour dire numéro de ligne, et j’ai dit que j'ai 3 lignes, je vais donc numéroter entre 1 et 3. Et pour chaque ligne, ce qu'il faut, c'est que j'affiche chaque colonne. Afficher une colonne, c'est afficher une étoile, donc je vais faire une boucle for et des colonnes (j'en ai 4) donc, ncol pour numéro de colonne (je vais aller de 1 jusqu'à 4) et à chaque fois, j'affiche (surtout sans passer à la ligne), une étoile, donc, je peux afficher le caractère étoile. C'est tout ce que je fais pour chaque colonne, et ensuite, quand j'ai terminé ce for, j'ai affiché mes 4 étoiles pour les 4 lignes. Et donc là, il faut que je passe à la ligne, donc je peux le faire par exemple en mettant un message vide, un message sans caractère, mais le ln nous fera passer à la ligne. Et voilà qui devrait marcher.

Je vais le sauver, le compiler, c'est bon. Et pour l'exécuter, j'agrandis un peu ma fenêtre. Voilà ! donc c'est bien le but que j'avais. Et alors, l'intérêt de ce programme, c'est que je peux très facilement changer le nombre de lignes, le nombre de colonnes. Il n'y a qu'une petite opération à faire, je re-sauve, je re-compile et, voyez, j'ai un rectangle plus grand.

Et je peux très facilement aussi avoir un programme qui s'adapte, c'est-à-dire, faire du numéro de ligne et du numéro de colonne des entrées du programme. Je peux demander : « Entrez le nombre de lignes ». Alors, il faut que je le déclare : ligne. Je le prends au clavier : ligne égal MOOC point readInt. Et là, maintenant, je vais remplacer le 5 par  ligne. Je sauve, je compile, j'exécute. Combien de lignes je veux ? J'en veux 4. Voilà, j'ai eu mes 4 lignes.

On va reprendre maintenant le fil de l'exposé, en se donnant maintenant pour but l'affichage d'un triangle. Et là, par rapport au rectangle, la différence, c'est que le nombre de colonnes dépend du numéro de ligne. Ici, si j'affiche les numéros de ligne et de colonne, je vois que ligne 1, j'ai une colonne. Ligne 2, deux colonnes. Et là donc, dans cet exemple, on va utiliser le numéro de tour des lignes dans la boucle, pour savoir combien de colonnes il y a. Je vais repasser sur la démo. (Pardon, petit problème technique), je vais changer mon programme de Rectangle en Triangle.

Donc, ce qui change ici, c'est que le numéro de colonne sera différent à chaque ligne, et le nombre de colonnes sera égal au numéro de ligne. Ce que je vais mettre ici, c'est nlig. Donc, au premier tour de boucle de la grande boucle, quand nlig vaut 1, je n'aurai qu'une colonne et ici, je vais aller jusqu'à 1. Quand nlig aura été incrémenté ici, nlig vaudra 2 j'aurai deux colonnes, etc.

Donc je vais faire un save as pour changer le nom de mon programme, qui s'appelle maintenant Triangle.java, le nom du fichier doit correspondre au nom du programme, je compile, et je vais faire cette exécution. Je vais demander 4 et voilà : j'ai quatre lignes et on voit bien que c'est un triangle.

Autre exemple que je ne traiterai pas jusqu'au bout : c'est la table de multiplication. Voyez, ici, le but, c'est de représenter les tables de multiplication dans un tableau, où dans chaque case, je mettrai le résultat de la multiplication du numéro de ligne par le numéro de colonne. Donc dans cette case, 1ère ligne, 3ème colonne, je mettrai une fois trois : 3. Donc la formule, c'est que dans chaque case, on met le numéro de ligne multiplié par le numéro de colonne. Je vais avoir ici, comme tout à l'heure (comme dans le rectangle), je vais avoir la multiplication, je vais avoir des lignes et des colonnes. Je vais supprimer le début du programme, parce qu'on va avoir à chaque fois, on ira jusqu'à 9. 9 lignes et 9 colonnes, c'est cela les tables de multiplication. Et ce qu'on va afficher là, à chaque fois, ce sera le numéro de ligne multiplié par le numéro de colonne. Et voyez, dans le calcul, dans le corps de ma boucle, je peux utiliser les deux numéros, le numéro de tour qui compte les tours de la première boucle for, et le numéro qui compte les tours de la deuxième boucle for. Je vais mettre un petit espace pour séparer les nombres, et cela doit être bon, je vais faire un save as pour changer le nom en Multiplication point java. Je compile, et je vais exécuter, voilà.

Donc, j'ai mes tables de multiplication et là où je ne vais pas jusqu'au bout, je vous ai annoncé que je n'irai pas jusqu'au bout de cet exemple, c'est que dans la mise en page, les colonnes ne sont pas bien alignées, donc il faudrait changer un petit peu les choses pour que l'affichage soit un peu plus joli. Mais cela sort un peu de notre sujet, qui était les boucles imbriquées et sur ce dernier exemple s'achève cette vidéo.

6.1

Bonjour.

Nous voilà déjà arrivés à la fin de notre cours. Et cette semaine, nous n'allons pas voir de nouvelles instructions Java, ni de nouvelles constructions du langage. Nous allons essayer de mieux utiliser ce que nous connaissons déjà.

Ce qu'on a vu le plus récemment, ce sont les tableaux, qui permettent de mettre plusieurs données du même type dans une même structure en mémoire, avec un système de numérotation qui permet de faire des opérations en boucle sur ces différentes données, notamment au moyen de la boucle for. Et on a vu donc, que pour créer un tableau, il faut faire un new. C'est le new qui crée un tableau, mais il reste des problèmes inhérents aux instructions Java qui traitent des tableaux.

Le premier problème, c'est que le nombre de données dans un programme peut varier au fil de l'exécution du programme, alors que la taille du tableau reste fixe.

Donc, le premier problème, des exemples. C'est par exemple, si on a un logiciel de gestion des employés d'une entreprise, on peut, au cours de l'exécution de ce programme, avoir à prendre en compte l'embauche d'un nouvel employé ou au contraire, le départ d'un employé parti sous d'autres cieux. Donc, comment on peut faire si on a un tableau de taille fixe et que le nombre, par exemple, de noms d’employés ou de numéros d'employés à stocker, varie au fil du temps?

Le deuxième problème, c'est qu'il faut souvent plusieurs valeurs pour caractériser une réalité. Il y a beaucoup de choses qu'on ne peut pas réduire à un nombre, ou un message, ou un caractère. Et pour beaucoup de choses qu'on désire manipuler, il faut rassembler des données de types différents comme par exemple, pour un employé : un nom, un matricule, une date de naissance, etc.

Donc les solutions : comme je vous le disais en introduction, on ne va pas introduire de nouvelles constructions Java. De toute façon, il n'y en a pas vraiment. Donc on va utiliser ce qu'on connaît déjà pour contourner les problèmes.

Pour le nombre de données variables, pour la taille qu'on voudrait faire varier, on verra deux techniques : soit créer un tableau à chaque changement de taille. Donc, chaque fois qu'on aura un élément en plus ou en moins, on crée un nouveau tableau à la nouvelle taille. Ou alors, l'autre technique consiste à utiliser un tableau trop grand, et à gérer les cases vides du tableau.

Pour les données de types différents, on n'a pas d'autre solution que d'utiliser des tableaux différents. Et là, la solution, c'est de coordonner plusieurs tableaux pour représenter une même réalité.

Donc, on est dans la dernière semaine du cours, et le programme de la semaine, ce sont les deux problèmes que je viens de vous présenter, et aussi, en guise de conclusion, un bref tour d'horizon de ce qui manque dans notre cours : d'une part, pour savoir programmer, donc ce qui est le thème précis du cours, mais aussi, pour savoir développer, puisque le développement de logiciel ne se réduit pas à la programmation. Donc, on fera un tour d'horizon de tout ce qui vous reste à voir si vous désirez aller plus loin, ou de tout ce qu'il faudrait voir pour avoir une idée complète de l'industrie de développement de logiciels.

6.2

Dans cette vidéo, nous allons voir comment adapter la taille d'un tableau aux besoins qu'on a dans une application donnée. Les tableaux ont une taille fixe, fixée à la création. Quand on fait un new pour créer un tableau, on donne la taille, et cette taille ne change jamais après. Mais on peut avoir un nombre de données qui change au fil du programme.

Par exemple, quand on a un tableau qui représente les employés d'une entreprise, et qu'il y a une embauche, le tableau n'est plus assez grand pour contenir tous les employés de l'entreprise. Ou alors, si on a les notes d'un élève et il a un nouveau contrôle, cela fait une note de plus. Donc, si on représente les résultats scolaires d'un élève, le nombre de notes change au fil du temps. Un autre exemple où c'est dans le sens inverse, c'est que si on a le classement d'une course, et qu’après coup, un coureur est disqualifié, il y a moins de coureurs dans le classement.

Donc on a deux solutions et demie au problème. La première solution, c'est de créer un nouveau tableau à chaque changement de taille. La deuxième, c'est de créer un tableau trop grand et gérer des cases vides. Et la demi-solution, c'est de mélanger les deux techniques pour résoudre le problème.

Alors prenons la première des solutions qui consiste à créer un nouveau tableau et supposons qu'on ait un tableau de taille 10. On veut stocker une donnée de plus. Donc les opérations à faire, c'est, premièrement, utiliser une nouvelle variable de type tableau, parce que, pendant un certain temps, on va avoir deux tableaux : celui de taille 10 et celui de taille 11. Et donc, on va avoir besoin de deux noms pour ces deux tableaux. Donc on a besoin d'une nouvelle variable pendant un certain temps et on crée un tableau de taille 11, et on l'affecte à cette variable. Ensuite, il faut recopier les 10 éléments de l'ancien tableau, celui de taille 10 dans celui de taille 11. Et on ajoute la donnée en plus dans la dernière case du tableau. Et dernière opération, c'est, le tableau de taille 11, on l'affecte au nom de l'ancienne variable.

Voyons un exemple : ici, donc, ce n'est pas une taille 10, c'est une taille 5. Le nom de notre variable, c'est tab. On voit qu'on a un tableau avec 5 valeurs, et il s'agit de rajouter la valeur 9.

Donc, première étape : créer une nouvelle variable, que j'appelle temp parce que c'est une variable qui n'a un usage que temporaire, et un tableau qui permet de stocker les 5 valeurs qu'on avait ici, et la 6ème qu'on a là. Donc, un tableau de 6 cases, numérotées de 0 à 6. Quand on crée un tableau, forcément, dans chaque case, ici, on a 0. Troisième étape, on recopie les valeurs de l'ancien tableau dans le nouveau et cela, cela suppose de faire une boucle pour recopier les valeurs une après l'autre. Voilà, on l'a recopié. Encore une opération : on rajoute le 9 dans la dernière case. Et la dernière chose, c'est : on fait une affectation de ce nouveau tableau à l'ancienne variable.

Et voilà comment on a pu mettre à jour nos données au moyen d'un nouveau tableau. Et là, j'ai mis une croix sur l'ancien tableau. L'ancien tableau, on n'en a plus besoin. Après, est-ce qu'il est vraiment supprimé ou pas ? On s'en fiche un peu, cela revient au même. La vérité, c'est qu'il ne sera pas forcément supprimé tout de suite, mais si un jour, le programme a besoin de mémoire, il supprimera ce tableau.

Une petite démo qui illustre cette chose-là. Donc, j'ai déjà préparé un début de programme. Le programme s'appelle Agrandir et j'ai créé un tableau de taille 5. Et dans chaque case, j'ai mis  i plus 1. Au moyen de la boucle, j'ai mis dans ce tableau : dans la case numéro 0, je mets 0 plus 1 : 1. Dans la case numéro 1, je mets 2, etc. Donc, dans les 5 cases, je vais avoir 1, 2, 3, 4, 5. Et mon but, c'est de rajouter 9 à la fin du tableau.

Première opération : créer un nouveau tableau avec une case de plus. Comme tout à l'heure, je l'appelle temp et je l'initialise donc à la bonne taille.

Ensuite, deuxième opération : il faut que je recopie les valeurs de l'ancien tableau dans le nouveau. Je fais cela au moyen d'une boucle : int num égal 0; num inférieur à 5; num égal num plus 1. Donc là, je vais parcourir toutes les cases de l'ancien tableau. C'est pour cela que je vais jusqu'à 5 et pas jusqu'à 6. Et dans temp de num, je vais mettre ce qu'il y avait dans le tableau à la case num.

Ensuite, dans la dernière case. La dernière case d'un tableau de taille 6 a le numéro 5, parce qu'on commence à numéroter à 0. Donc, dans temp de 5, je vais mettre la valeur 9 que je voulais rajouter. Et j'ai terminé de créer le tableau avec sa case de plus et la donnée que je voulais rajouter.

Il faut que je fasse maintenant une affectation, donc tab égal temp. Ce que j'ai mis à la fin, ici, c'est juste une boucle pour afficher mon tableau, pour m'assurer que cela fait bien ce que je souhaitais.

Je sauve, je compile, j'agrandis la fenêtre, et j'exécute. Voilà. 1, 2, 3, 4, 5, 9, j'ai réussi à rajouter une valeur dans le tableau.

Voyons maintenant… ce qu'il faut penser de cette technique. Cela marche. C'est coûteux, parce que, à chaque fois, il faut créer le tableau, recopier, et donc, c'est à éviter sur les gros tableaux, et à éviter aussi si le nombre de données varie souvent. Il faut éviter de faire cela trop souvent. Et dans ce cas-là, il vaut mieux utiliser la deuxième technique que je vais vous présenter maintenant, qui consiste à gérer des cases vides.

Donc, au lieu de faire un tableau adapté à la taille des données, on prend un tableau trop grand en se disant : « Bon, je vais garder des cases vides, et éventuellement, elles se rempliront au fil du temps ». Donc certaines contiennent une donnée, d'autres ne contiennent pas de données. Le problème, c'est qu'une case de tableau ne peut pas être vide. C'est-à-dire, dans la case du tableau, il y a nécessairement quelque chose. Le vide n'existe pas. Par exemple, sur un tableau d'entiers, dans chaque case, il y a un entier, quoi qu'on fasse. Et le vide n'est pas un entier. Donc la case n'est pas vraiment vide, c'est le programmeur et le programme, qui doivent savoir quelle case est vide et quelle case est pleine, et ne jamais garder des cases vides.

Donc il y a plusieurs techniques pour gérer les cases vides. Je vous propose d'en voir une, c'est de séparer en deux zones le tableau avec, d'un côté les cases occupées, je vais les mettre à gauche, de l'autre côté, les cases vides, je les laisserai à droite, et stocker dans une variable de type int la limite entre les deux zones du tableau. Et en fait, je vais mettre le numéro de la première case vide qui est aussi le nombre de cases occupées, vu qu'on numérote à partir de 0. Et quand on ajoute une donnée, on la mettra dans la première case vide et on augmentera ce nombre de cases occupées, ou le numéro de la première case vide. Quand on enlève une donnée, c'est le contraire, il faut bien libérer la case la plus à droite du tableau.

Donc là, j'ai représenté en rose la partie occupée du tableau, en jaune la partie libre. Donc on voit que la première case libre, c'est la case numéro n. Donc j'ai une variable ici, que j'appelle  nb, et qui contient n. Si je veux ajouter une donnée, la donnée que je veux ajouter, supposons que je veuille ajouter 9, je vais la mettre dans cette case-là. Et donc, il y aura une case occupée de plus dans mon tableau. (C'est un 9, c'est mal écrit, mais ici, c'est censé être un 9). La première case vide du tableau, ce n'est plus n, ou le nombre de cases occupées, ce n'est plus n, c'est n plus 1. Donc, chaque fois que je rajoute une donnée, je la mets ici. S'il y en une de plus, je la mettrai là, s'il y en a encore une, etc. Et à chaque fois, j'augmente ici n plus 1,  n plus 2, etc.

Maintenant, si j'enlève une donnée. Si j'enlève une donnée, la case à libérer, c'est celle-là. Mais la donnée que j'avais à enlever n'est peut-être pas celle qui était dans cette case-là. Peut-être que je voulais enlever un 9 qui était ici, dans la case numéro 2. Alors, comment je vais faire ? Ce que je vais faire, c'est que cette donnée que j'ai là, je vais la recopier ici, à la place de celle que je veux enlever. Et ici, je vais lire n moins 1. Donc je n'enlève pas vraiment la donnée de la case n moins 1, mais je considère que la case n moins 1 est libre. De cette façon, je peux gérer les cases vides.

Donc, gérer les cases vides, c'est peut-être conceptuellement un peu plus compliqué. Cela demande de ne pas se tromper, mais c'est souvent préférable. En revanche, même avec cette technique, il se peut que, arrivé à un moment, on n'ait plus de case vide. Le tableau est plein, et, dans ce cas-là, il faut utiliser la première technique : créer un nouveau tableau plus grand. mais on ne prendra pas seulement plus grand d'une case, mais de 10 cases, de 20 cases, de 30 cases, pour pouvoir avoir un peu de marge, et pouvoir durer plus longtemps de cette façon.

Voilà les deux techniques pour essayer d'adapter la taille d'un tableau à nos données.

6.4

Dans cette vidéo, nous allons voir comment représenter des données complexes quand les données qu'on veut représenter ne se limitent pas à un nombre ou un caractère.

On a en Java des types élémentaires comme int, double, char, boolean. Le type string est un type un peu plus compliqué. C'est une suite de plusieurs caractères, donc plusieurs chars. Et les tableaux sont des suites de plusieurs valeurs, mais du même type. Donc, on a des types élémentaires qui sont les nombres, les caractères, les booléens, et des types, que ce soit string ou les tableaux, où on a une suite de plusieurs valeurs du même type.

Mais certaines données n'appartiennent à aucun de ces types par nature. Et par exemple, si on veut représenter des personnes, ce ne sont pas des nombres, ce ne sont pas des entiers, ce ne sont pas des booléens. Les produits, c'est pareil, cela ne se réduit pas à un nombre, ou à un nom, les animaux, etc. Les problèmes à résoudre, ce pour quoi on veut utiliser des programmes, sont souvent des réalités complexes, donc typiquement, des personnes, les lecteurs d'une bibliothèque, les employés d'une entreprise, etc. C'est ce genre de réalité qu'on veut représenter dans nos applications. On ne peut pas représenter la totalité de données complexes dans un programme, on est obligé d'utiliser une abstraction où on représente une réalité complexe au moyen de quelques attributs qui, eux, sont des données simples d'un des types qu'on connaît déjà.

Une personne, cela a plusieurs facettes, et ce ne seront pas les mêmes données qu'on emploiera si on le considère comme un employé, s'il s'agit par exemple de la paie, d'un assuré social, s'il s'agit de rembourser des choses, d'un consommateur, donc s'il s'agit de fidéliser les clients, un citoyen, s'il s'agit de cartes électorales ou de choses de ce genre, membre d'une famille, lecteurs d'une bibliothèque. A chaque fois, on utilisera des attributs différents selon l'application.

Donc pour une personne, le genre d'attributs, ce sera le nom et le prénom, l'âge, le numéro NC, le numéro de Sécu, et je mets un point d'interrogation, parce que le numéro de Sécu, s'il s'agit des lecteurs d'une bibliothèque, on n'a pas besoin du numéro de Sécu. L'adresse, si c'est un lecteur de bibliothèque, on a besoin de l'adresse. Donc ce sont plusieurs données de type différent, et on ne peut pas les mettre dans un tableau, pour cette raison. Parce qu'on ne peut pas mettre par exemple, un nom et un âge dans le même tableau. Ce n’est pas de même type.

Donc pour notre exemple là, on va considérer que les attributs qui nous intéressent pour une application, sont le nom, le prénom et l'année de naissance. Donc pour une seule personne, et s'il y a une seule personne dans le programme, on prend 3 variables : 2, le nom et le prénom, sont des string, et l'année de naissance, c'est un entier. Maintenant, s'il y a plusieurs personnes à représenter, on va avoir besoin de 3 tableaux : un tableau pour les noms, un tableau pour les prénoms, et un tableau pour les années de naissance. Et il faudra coordonner pour savoir retrouver les informations qui concernent une personne dans un tableau, dans les 3 tableaux différents. Donc par exemple, là, j'ai pris 3 personnes différentes. Vous voyez que si je veux par exemple les informations qui concernent Victor Hugo, il faut que j'aille les chercher dans 3 tableaux différents qui sont dans 3 variables différentes : nom, prénom, et naissance, ce sont des noms de variable. Il faut que j'aille chercher dans la case 0 du premier tableau, la case 0 du deuxième tableau, la case 0 du troisième tableau.

Donc là, en fait, vous avez compris que ce qui nous permet de coordonner les tableaux, c'est le numéro de case. Je sais que dans chaque tableau, pour un numéro de case donné, j'aurai les informations qui concernent une personne. Donc je vais avoir comme cela mes personnes, qui sont représentées au moyen de variables différentes, mais je vais savoir retrouver les informations qui les concernent facilement, en utilisant ce numéro qui est une clé d'accès. Chaque personne aura son numéro.

Je vais faire maintenant une démo avec un autre exemple : un stock de produits. J'ai basculé un peu trop vite dans le mode démo, je voulais vous présenter un peu ce que j'allais faire dans cette démo. Il y aura donc 4 attributs : intitulé, référence, prix et quantité. Je vais faire un petit programme, où je vais rentrer d'abord des produits, et ensuite je ferai deux opérations : afficher les informations concernant un produit, et augmenter le prix d'un produit de 10 %.

Voilà donc ce que je vais faire dans cette démo. J'ai 4 attributs, j'ai besoin de 4 tableaux. La référence, je vais dire que c'est un entier : je vais l'appeler ref. Et puis, pour que ce ne soit pas trop long, je vais rentrer seulement 3 articles, mais vous comprenez bien que cela pourra se généraliser à 100 ou 1000 articles sans problème. L'intitulé, je ne peux pas l'appeler int, parce que…, je vais l'appeler nom. Je prends évidemment des tableaux de la même taille, puisque, si j'ai n produits, il me faudra n cases dans chacun des tableaux. Le prix, cela va être un double et la quantité, cela va être un entier. Je vais l'appeler qte. J'ai mes 3 tableaux. Maintenant, il faut les remplir. Je vais saisir au clavier les informations. Num égal 0; num plus petit que 3; num égal num plus 1. Donc je vais mettre dans la case numéro i du tableau des références ce que je vais lire au clavier : readInt. C'est un petit peu long, il faut que je demande à chaque chose… Ce n'est pas ref, c'est nom de i égal MOOC point readString ». Et puis, pour aller plus vite, je vais mettre le prix directement, je vais l'inventer. J'ai mis i, alors, ce n'est pas i, c'est num le numéro de la case. Le prix, je vais mettre : 10 fois num plus 10, c'est complètement arbitraire. Et la quantité de chaque produit va être de num fois 5, là aussi c'est complètement arbitraire. C'est juste pour gagner un peu de temps à l'exécution. Maintenant, je vais demander à l'utilisateur quel produit il veut afficher d'une part, et augmenter le prix, d'autre part. Je vais déclarer pour cela une nouvelle variable, que je vais appeler laref, et je demande la référence du produit auquel on s'intéresse : laref égal MOOC point readInt, et je vais commencer par afficher ce produit. Et afficher ce produit, cela veut dire afficher toutes les informations, pour aller plus vite, je vais utiliser une concaténation donc, je vais commencer par mettre laref (je peux mettre laref, je sais que c'est la même)... Il faut que je le cherche d'abord, ce produit. Donc, je vais chercher son numéro de case, je vais initialiser cela à 0. Et je vais faire un while : tant que je ne trouve pas dans le tableau la référence que je cherche, donc dans le tableau de références : je vais le parcourir et tant que je ne le trouve pas, je vais essayer la case suivante : nocase plus un. Quand je sors de la boucle while, j'ai trouvé. Donc, c'est « tant que c'est différent », pas « tant que c'est égal ». Tant que je n'ai pas, dans mon tableau, trouvé la référence recherchée, je vais voir la case suivante en ajoutant 1 au numéro de case. Quand je sors de cela, j'ai dans nocase le numéro de la référence recherchée.

Donc, je peux commencer par afficher la référence, plus un espace, plus nom de nocase. J'ai trouvé la référence dans la case numéro nocase. Et donc, je trouverai le nom dans le tableau nom, au même numéro de case. Le prix, je vais afficher un petit message pour bien dire que c'est le prix, prix de nocase. Quantité : plus qte de nocase . Voilà l'affichage des informations concernant le produit. Et maintenant, je veux augmenter le prix de 10 %. Prix de nocase égal prix de nocase fois 90 point 0 divisé par 10 point 0 ». Et voilà. Et peut-être, je ré-affiche une nouvelle fois mon produit, pour voir que le prix a augmenté.

Voilà mon programme. Je le sauve, je le compile. Cela m'aurait étonné qu'il n'y ait pas d'erreur. Alors, c'est quoi? Il y a un plus de trop. Voilà. Et on va maintenant exécuter ce programme. Alors, la référence 100 : cela va être la brosse à dents. La référence 200 : les croquettes, et la référence 300 : le balai. Alors, là, il me demande quel produit je veux. Eh bien, je vais prendre par exemple les croquettes. Oh la la ! On voit que je me suis trompé : je voulais augmenter le prix, cela a diminué. Ah zut, je n'ai pas fait la bonne opération : j'ai fait   moins 10 %  au lieu de faire plus 10 % . Ecoutez, ce n'est pas grave, je pense que vous saurez corriger de vous-même. Et tant mieux pour le consommateur.

Voilà qui conclut cet exposé sur la façon de coordonner plusieurs tableaux.