319 – Concevoir et implémenter des applications

Rapport personnel

Date de création : xx.xx.2024  
Version x du 28.11.2024

|  |  |  |
| --- | --- | --- |
|  |  | Module du 22.08.2024 au 16.01.2025 |

Table des matières

Table des matières

[1 Les bases de Java 4](#_Toc183720593)

[1.1 La méthode main (), utilité et sa déclaration exacte 4](#_Toc183720594)

[1.2 Les 8 types de base en java 4](#_Toc183720595)

[1.3 Déclaration d’une variable et affectation d’une variable 5](#_Toc183720596)

[1.4 Déclaration d’une constante 5](#_Toc183720597)

[1.5 Les commentaires (une ligne ou plusieurs lignes) 5](#_Toc183720598)

[1.6 L'écriture sur la console avec la méthode sout (println et print) 5](#_Toc183720599)

[1.7 Les opérateurs en Java 6](#_Toc183720600)

[1.7.1 Les opérateurs de calcul (+ - \* / %) 6](#_Toc183720601)

[1.7.2 Les opérateurs d'assignation (= += -= \*= /=) 6](#_Toc183720602)

[1.7.3 Les opérateurs d'incrémentation et décrémentation (++ --) 7](#_Toc183720603)

[1.7.4 Les opérateurs de comparaison (== < > <= >= !=) 7](#_Toc183720604)

[1.7.5 Les opérateurs logiques (|| && ! ^) 8](#_Toc183720605)

[1.8 Les conditions 9](#_Toc183720606)

[1.8.1 if / if else /else 9](#_Toc183720607)

[1.8.2 switch case default break 9](#_Toc183720608)

[1.9 Nombres aléatoires 10](#_Toc183720609)

[1.9.1 La méthode Math.random() 10](#_Toc183720610)

[1.9.2 La génération correcte d’un nombre entier aléatoire entre deux limites 10](#_Toc183720611)

[1.10 Les boucles 10](#_Toc183720612)

[1.10.1 For 10](#_Toc183720613)

[1.10.2 While 10](#_Toc183720614)

[1.10.3 Do-while 11](#_Toc183720615)

[1.10.4 Les mots-clé break et continue 11](#_Toc183720616)

[1.11 Les tableaux 12](#_Toc183720617)

[1.11.1 Déclaration et création et taille d’un tableau 12](#_Toc183720618)

[1.11.2 Remplir un tableau avec une valeur 12](#_Toc183720619)

[1.11.3 Lire et écrire dans un tableau 12](#_Toc183720620)

[1.12 Les méthodes 13](#_Toc183720621)

[1.12.1 Déclaration, paramètres et type de retour 13](#_Toc183720622)

[1.12.2 Retourner une valeur 13](#_Toc183720623)

[1.12.3 Appel d’une méthode et récupération de sa valeur de retour 13](#_Toc183720624)

[1.13 Les chaînes de caractères 14](#_Toc183720625)

[1.13.1 La classe String 14](#_Toc183720626)

[2 Algorithmique de base 15](#_Toc183720627)

[2.1 Code qui teste si le nombre est positif ou négatif 15](#_Toc183720628)

[2.2 Code qui test si le nombre est pair ou impair 15](#_Toc183720629)

[2.3 Code qui échange des valeurs de 2 Integer 16](#_Toc183720630)

[2.4 Remplir un tableau avec une même valeur 17](#_Toc183720631)

[2.5 Rechercher la position de la première occurrence d’une valeur dans un tableau 17](#_Toc183720632)

[2.6 Rechercher la position de la dernière occurrence d’une valeur dans un tableau 17](#_Toc183720633)

[2.7 Remplacer une valeur par une autre dans un tableau 19](#_Toc183720634)

[2.8 Compter le nombre d’occurrence d’une valeur dans un tableau 19](#_Toc183720635)

[2.9 Trouver la plus grande valeur contenue dans un tableau 21](#_Toc183720636)

[2.10 Trouver la somme des valeurs d’un tableau 21](#_Toc183720637)

[2.11 Calculer la moyenne des valeurs contenues dans un tableau 21](#_Toc183720638)

[2.12 Remplir un tableau avec des valeurs aléatoires 22](#_Toc183720639)

[2.13 Mesurer la taille d’un tableau 22](#_Toc183720640)

[2.14 Inverser le contenu d’un tableau 22](#_Toc183720641)

[3 Structogrammes 23](#_Toc183720642)

[3.1 C’est quoi, utilité ? 23](#_Toc183720643)

[3.2 Représentation visuelle des opérations 23](#_Toc183720644)

[4 Conclusions 26](#_Toc183720645)

[4.1 Ce que j’ai apprécié 26](#_Toc183720646)

[4.2 Ce que j’ai moins apprécié 26](#_Toc183720647)

[4.3 Mon auto-évaluation 26](#_Toc183720648)

[4.4 Conclusion 26](#_Toc183720649)

# Les bases de Java

## La méthode main (), utilité et sa déclaration exacte

public static void main(String[] args) {

// Code du programme ici

}

* Méthode pour commencer l'exécution du programme.
* Sans une méthode main() le programme ne s'exécutera pas.

## Les 8 types de base en java

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Primitive** |  | **Signification** | **Taille** |  | **Plage de valeurs acceptée** |
| char |  | Caractère | 16 bits |  | valeur du jeu de caractères Unicode (65000 caractères possibles) |
| byte |  | Entier très court | 8 bits |  | -128 à 127 |
| short |  | Entier court | 16 bits |  | -32’768 à 32’767 |
| int |  | Entier | 32 bits |  | -2^31 à 2^31-1 (-2’147’483’648 à 2’147’483’647) |
| long |  | Entier long | 64 bits |  | -2^63 à 2^63-1 (-9’223’372’036’854’775’808 à 9’223’372’036’854’775’807) |
| float |  | flottant (réel) | 32 bits |  | -1.4\*10-45 à 3.4\*1038 |
| double |  | flottant double | 64 bits |  | 4.9\*10-324 à 1.7\*10308 |
| boolean |  | booléen | (1 bit serait suffisant,  mais stocké sur 8 bits) |  | **true** ou **false** |

String n’est pas un type de base

## Déclaration d’une variable et affectation d’une variable

Pour déclarer une variable, on écrit

Int variable

Pour affecter une variable, on écrit

variable = valeur1

on peut tout faire sur une ligne

int variable = valeur1

## Déclaration d’une constante

public class JavaApplication82 {

public final static double PI = 3.1415926;

public static void main( String[] args ) {

System.out.println( "La valeur de pi est " + PI );

}

}

## Les commentaires (une ligne ou plusieurs lignes)

Une ligne : On met // devant le texte à mettre en commentaire.

Plusieurs lignes : on met /\* *texte en commentaire* \*/ pour mettre en commentaire sur plusieurs lignes

## L'écriture sur la console avec la méthode sout (println et print)

Sout : est l’abréviation de System.out.println(). C’est pour afficher quelque chose dans la console.

Pour mettre Système.out.print() on supprime le ln

La différence est que sans ln c’est quand on ne veut pas de retour à la ligne

## Les opérateurs en Java

### Les opérateurs de calcul (+ - \* / %)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opérateur** | **Dénomination** | **Effet** | **Exemple** | **Résultat (int x=7)** |
| + | opérateur d'addition | Ajoute deux valeurs | x+3 | 10 |
| - | opérateur de soustraction | Soustrait deux valeurs | x-3 | 4 |
| \* | opérateur de multiplication | Multiplie deux valeurs | x\*3 | 21 |
| / | opérateur de division | Calcul le quotient de la division de deux valeurs | x/3 | 2 |
| % | opérateur de modulo | Calcul le reste de la division de deux valeurs | x%3 | 1 |
| = | opérateur d'affectation | Affecte une valeur à une variable | x=3 | Met la valeur 3 dans la variable x |
|  |  |  |  |  |

### Les opérateurs d'assignation (= += -= \*= /=)

|  |  |
| --- | --- |
| **Opérateur** | **Effet** |
| += | addition deux valeurs et stocke le résultat dans la variable (à gauche) |
| -= | soustrait deux valeurs et stocke le résultat dans la variable |
| \*= | multiplie deux valeurs et stocke le résultat dans la variable |
| /= | divise deux valeurs et stocke le quotient dans la variable |
| %= | divise deux valeurs et stocke le reste dans la variable |

### Les opérateurs d'incrémentation et décrémentation (++ --)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opérateur** | **Dénomination** | **Effet** | **Syntaxe** | **Résultat (int x=7)** |
| ++ | Incrémentation | Augmente d'une unité la variable | x++ ou ++x | 8 |
| -- | Décrémentation | Diminue d'une unité la variable | x-- ou --x | 6 |

### Les opérateurs de comparaison (== < > <= >= !=)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opérateur** | **Dénomination** | **Effet** | **Exemple** | **Résultat** |
| ==  **À ne pas confondre avec le signe d'affectation =** | opérateur d'égalité | Compare deux valeurs et vérifie leur égalité | x==3 | Retourne *true* si x est égal à 3, sinon *false* |
| < | opérateur d'infériorité stricte | Vérifie qu'une variable est strictement inférieure à une valeur | x<3 | Retourne *true* si x est inférieur à 3, sinon *false* |
| <= | opérateur d'infériorité | Vérifie qu'une variable est inférieure ou égale à une valeur | x<=3 | Retourne *true* si x est inférieur ou égal à 3, sinon *false* |
| > | opérateur de supériorité stricte | Vérifie qu'une variable est strictement supérieure à une valeur | x>3 | Retourne *true* si x est supérieur à 3, sinon *false* |
| >= | opérateur de supériorité | Vérifie qu'une variable est supérieure ou égale à une valeur | x>=3 | Retourne *true* si x est supérieur ou égal à 3, sinon *false* |
| != | opérateur de différence | Vérifie qu'une variable est différente d'une valeur | x!=3 | Retourne *true* si x est différent de 3, sinon *false* |

### Les opérateurs logiques (|| && ! ^)

|  |  |  |  |
| --- | --- | --- | --- |
| **Opérateur** | **Dénomination** | **Effet** | **Syntaxe** |
| || | OU logique | Retourne *true* si au moins une des deux conditions vaut *true* (ou *false* sinon) | condition1 || condition2 |
| && | ET logique | Retourne *true* si les deux conditions valent *true*(ou *false* sinon) | condition1 && condition2 |
| ! | NON logique | Retourne *true* si la variable vaut *false*, et *false* si elle vaut *true*) | !condition |

## Les conditions

### if / if else /else

if : if (condition){

instruction1

}

else : if (condition){

intrusction 1

...

}else if {

Instruction 2

...

}

else if : if (condition1) {

Instruction1

}else if (condition2){

Instruction 2

}

### switch case default break

switch (expression){

case valeur1 :

instructions;

break;

case valeur2 :

instructions;

break;

default :

instructions;

}

## Nombres aléatoires

### La méthode Math.random()

int nombre = ( int ) ( Math.random() \* ( MAX - MIN + 1 ) ) + MIN;

### La génération correcte d’un nombre entier aléatoire entre deux limites

Int nombre = (int) (Math.random() \* (MAX – MIN +1) ) + MIN

## Les boucles

### For

for (valeurDépart; conditionFin; incrément){

…

}

Exemple :

for (int i=0; i<5; i++){

System.out.println("i: " + i);

}

### While

while (condition){

…

}

Exemple : boolean uneCondition = true;

while (uneCondition){

…

if(…){

uneCondition = false;

}

}

### Do-while

do {

…

} while (condition);

Exemple :

boolean uneCondition = true;

do {

…

if(…){

uneCondition = false;

}

…

} while (uneCondition);

### Les mots-clé break et continue

* Sortir prématurément d’une itération: **continue**

**Exemple :** for (int i=0; i<5; i++){

…

if(…){

continue;

// Après cette instruction, on recommence une nouvelle itération.

}

…

}

* Terminer prématurément la boucle: **break**

**Exemple :** for (int i=0; i<5; i++){

…

if(…){

break;

// Après cette instruction, on quitte la boucle.

}

…

}

## Les tableaux

### Déclaration et création et taille d’un tableau

Déclarer : Int [] monTableau ;

Création sans valeurs : monTableau = new int[5];

Création avec valeurs : monTableau = new int[]{45, 23, 4};

### Remplir un tableau avec une valeur

tab[1] = 12;

### Lire et écrire dans un tableau

Lire :

int valeurCellule1 = tab[1];

System.out.println(valeurCellule1);

Connaître la taille d’un tableau :

int taille = tab.length;

sout(taille);

lire toutes les cellules d’un tableau :

for(int i=0; i < tab.length; i++){

sout(tab[i]);

}

## Les méthodes

* Une méthode est un bloc d’instructions appelé depuis une autre méthode (le main par exemple).
* On appelle une méthode avec son nom.

### Déclaration, paramètres et type de retour

Déclaration : public int addition(int a, int b) {

return a + b;

}

Paramètres : (Ils permettent de passer des informations nécessaires à l'exécution de cette méthode.)

public int addition(int a, int b) {

return a + b; // Renvoie la somme de deux entiers

}

Types de retour : indique ce que la méthode renvoie une fois exécutée.

### Retourner une valeur

Retourner une valeur est très utile. Ia ligne return vas être mis à la fin de la méthode. La variable après le mot clés return doit être du même type que la méthode

### Appel d’une méthode et récupération de sa valeur de retour

int nombre1 = 31

int nombre2 = 12

int nombre3 = 6

int somme = somme (nombre1, nombre2, nombre3) ;

## Les chaînes de caractères

### La classe String

Elle sert à pouvoir mettre du texte dans une variable. Grace à sa on peut directement demander à quelqu’un de rentrer du texte via un scanner et l’afficher quelque part.

# Algorithmique de base

Méthodes/bout de code qui feront certaine choses pratique pour la E1.

## Code qui teste si le nombre est positif ou négatif

public static boolean estPositif(int nombre){

boolean estPositif;

if (nombre >= 0) {

estPositif = true;

} else {

estPositif = false;

}

return estPositif;

}

## Code qui test si le nombre est pair ou impair

public static boolean estPair(int nombre){

boolean estPair;

if (nombre % 2 == 0) {

estPair = true;

} else {

estPair = false;

}

}

## Code qui échange des valeurs de 2 Integer

public static void inversement(int nombre1, int nombre2){

int nombre3;

nombre3 = nombre1;

nombre1 = nombre2;

nombre2 = nombre3;

System.out.println(nombre1);

System.out.println(nombre2);

}

## Remplir un tableau avec une même valeur

public static int[] inversement(int[] tableauARemplir, int nombre){

for (int i = 0; i < tableauARemplir.length; i++) {

tableauARemplir[i] = nombre;

}

return tableauARemplir;

}

## Rechercher la position de la première occurrence d’une valeur dans un tableau

public static int occurenceDeb(int[] tableauRecherche, int valeurAChercher){

int indexTrouver = -1;

for (int i = 0; i < tableauRecherche.length; i++) {

if (tableauRecherche[i] == valeurAChercher) {

indexTrouver = i;

break;

}

}

return indexTrouver;

}

## Rechercher la position de la dernière occurrence d’une valeur dans un tableau

public static int occurenceFin(int[] tableauRecherche, int valeurAChercher) {

int indexTrouver = -1;

for (int i = 0; i < tableauRecherche.length; i++) {

if (tableauRecherche[i] == valeurAChercher) {

indexTrouver = i;

}

}

return indexTrouver;

}

## Remplacer une valeur par une autre dans un tableau

public static int[] remplaceVal(int[] tableau, int valeurAChercher, int valeurRemplacement) {

for (int i = 0; i < tableau.length; i++) {

if (tableau[i] == valeurAChercher) {

tableau[i] = valeurRemplacement;

}

}

return tableau;

}

## Compter le nombre d’occurrence d’une valeur dans un tableau

public static int nbrFrequence(int[] tableauRecherche, int valeurChercher) {

int compteur = 0;

for (int i = 0; i < tableauRecherche.length; i++) {

if (tableauRecherche[i] == valeurChercher) {

compteur++;

}

}

return compteur;

}

public static int tabMin(int[] tableauRecherche) {

int min = tableauRecherche[0];

for (int i = 1; i < tableauRecherche.length; i++) {

if (tableauRecherche[i] < min) {

min = tableauRecherche[i];

}

}

return min;

}

## Trouver la plus grande valeur contenue dans un tableau

public static int tabMax(int[] tableauRecherche) {

int max = tableauRecherche[0];

for (int i = 1; i < tableauRecherche.length; i++) {

if (tableauRecherche[i] > max) {

max = tableauRecherche[i];

}

}

return max;

}

## Trouver la somme des valeurs d’un tableau

public static int sommeTab(int[] tableauCalcule) {

int somme = 0;

for (int i = 0; i < tableauCalcule.length; i++) {

somme = somme + tableauCalcule[i];

}

return somme;

}

## Calculer la moyenne des valeurs contenues dans un tableau

public static double moyenneTab(int[] tableauCalcule) {

double moyenne = 0;

for (int i = 0; i < tableauCalcule.length; i++) {

moyenne = moyenne + tableauCalcule[i];

}

moyenne = ( double ) moyenne / tableau.length;

return moyenne;

}

## Remplir un tableau avec des valeurs aléatoires

public static int[] valeurAléatoir(int[] tableauARemplir) {

for (int i = 0; i < tableauARemplir.length; i++) {

tableauARemplir[i] = (int) (Math.random() \* (MAX - MIN + 1)) + MIN;

}

return tableauARemplir;

}

## Mesurer la taille d’un tableau

public static int taille(int[] tableauMesure) {

int taille = 0;

for (int i = 0; i < tableauMesure.length; i++) {

taille++;

}

return taille;

}

## Inverser le contenu d’un tableau

public static int[] inverseLeTableau(int[] tableau) {

int[] tableauInverse = new int[tableau.length];

for (int i = 0; i < tableau.length; i++) {

tableauInverse[i] = tableau[tableau.length - 1 - i];

}

return tableauInverse;

}

# Structogrammes

se lit du haut vers le bas

## C’est quoi, utilité ?

Il permet de visualiser la logique d'un programme

On peut utiliser un structogramme pour décomposer un problème complexe en étapes simples

Les structogrammes sont plus simples et plus facile à lire que le code brut.

## Représentation visuelle des opérations

Les tests «if» :

Une image contenant texte, ligne, Police, blanc

Description générée automatiquement

Les tests de types «switch» :

Une image contenant texte, ligne, diagramme, capture d’écran

Description générée automatiquement

La boucle «for» :

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

La boucle «while» :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

La boucle «do-while» :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Exemples (pour points bonus) :

Une image contenant texte, capture d’écran, nombre, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

# Conclusions

## Ce que j’ai apprécié

Ce que j’ai apprécié dans ce module c’est voir son code se construire et après marcher. Réaliser ça de ses propres mains

## Ce que j’ai moins apprécié

C’est beaucoup de choses à retenir pour aller plus vite. Mais ce que j’ai le moins aimé c’est les tableaux

## Mon auto-évaluation

Je pense que je m’en sors plutôt bien, mais je dois quand même bien réviser.

J’ai encore quelque notion à apprendre

## Conclusion

En conclusion, j’ai appris les bases de la programmation. On à pu voire la déclaration de variables, les tableaux, les méthodes et bien d’autres choses. J’ai apprécié ce module.