JAVA – Helsinki University

Partie I

TOUJOURS coder une ligne et tester ! Couper le programme en plusieures parties.

1/ Printing et lire des entrées

1. Parametres : les informations qu'on met entre parathenses ; ce qui doit etre executer

2. Single line comment : //. Multi ligne commentaire : /* ….*/

3. Pour commencer input : Scanner scanner = new Scanner(System.in); Importation de l'outil input : import java.util.Scanner

4. Pour lire input : scanner.nextLine()

2/ Variables

5. Une variable contient l'information d'un type spécifique et a un nom (Exemple : String message, int numero, etc..). On peut lui accorder une valeur

6. Concatenation : usage de "+". Utile pour additioner une variable et un string etc.

7. Pas besoin de declarer une deuxieme fois le type d'une meme variable quand on veut changer sa valeur : int valeur = 3. Lors du changement : valeur = 4 directement

8. Une variable ne peut avoir qu'une seule valeur

9. On ne peut pas changer la valeur d'un type de variable. Un boolean qui prend un integer etc .. Exception : un int affecté à un double.

3/Calculs

10. Lors de la lecture du input de user, il faut convertir la reponse du string en integer, double, directement dans le scanner.nextLine.

- Integer.valueOf
- Double.valueOf
- Boolean.valueOf

Exemple : int valeur = Integer.valueOf(scanner.nextLine()).

11. Pour convertir un int (fisrt et second) on met (double) devant. Dans une division on peut pas mettre (double)(first/second) car la division se fait avant la conversion. Donc 3/2 = 1 puir conversion 1.0. Le resultat d'une division assignée à un variable de type int aura un resultat en int. On peut convertir en multipliant avec 1.0 l'expression mathematique.

4/Déclarations et opérations conditionnelles

12. Quand le conditional statement est “true”, la commande qui suit est executée. Si la commande est “false”, la commande est sautée et on passe à la commande d’après.

13. Indentation dans TMC : alt shift F

14. else if = quand on a de multiples conditions

15. Pour trouver le restant d’une division on utilise %

16. Pour comparer des strings on utilise ‘equals’ xxxxx.equals (“”) avec xxxx etant la le nom de la variable.

17. And &&, Or ||, not !

18. Dans les conditions, on commence par écrire celle qui est la plus “demandeuse” (on met les exceptions en premier)

19. Pour utiliser Math.sqrt (chiffre au carré), on est obligé d’utilisé un double.
Exemple : double carre = Math.sqrt(calcul)

20. Les variables comme Scanner reader sont définies avant une loop. Les variables comme les valeurs de l’utilisateur qui sont spécifiques à la loop sont définies à l’intérieur de la loop.

Partie II

1/Encore plus de boucles

21. Loop de type : while (true) {condition → break) {recommence tant que la condition n’est pas remplie}

22. Loop de type continue : while (true) { condition ‘continue’ } {condition ‘break’ (sinon le programme est infini si on met pas ‘break)}

23. Si je veux utiliser une variable après une loop, il faut que je l’introduise avant la loop. Sinon je peux pas y accéder.

24. Incrémentation +1 s’écrit xxxxx++

25. Loop for : for (int i=0; i<x;i++) (sout(i)) for (introduire une variable, condition, augmenter/diminuer/changer le compteur)

26. Raccourci de (x = x +1) → (x+=1)

27. For loop est interessante quand le nombre est predéfini par le user. La while loop est interessante jusqu’à ce que le user entre un nombre bien précis.

28. Structure clean d’une loop
// création de variables necessaires dans la loop

```
while (true) {
    // lecture de input
    // arreter la loop – break
   // checker des inputs invalids – continue
        // ce qu'on fait avec le input valid
         }
         fonctionnalités à exécuter quand la loop se finit
```

2/Méthodes

29. Créer une méthode : public static void method(type nomVariable, type nomVariable2) {
    fonctionnalités
    }. Appeler une methode directement dans le main : method();

30. On y ajoute les parametres en parentheses. Pour appeler dans le main la methode(int chiffre) => methode(42554);

31. Les variables d'une methode peuvent avoir le meme nom que la variable d'une autre tout en étant distinctes. Elles sont indépendantes.

32. Public static void methodRetourneAucuneValeur()
    Public static int methodeRetourneUnInt() Exemple : return 10;
    Public static double methodeRetourneUnDouble() Exemple: retourne 12.1 ou une expression mathematique.

33. La commande return sans aucune valeur arrete l'execution du code

34. On ne peut pas acceder à partir du main aux variables d'une methode. On ne peut acceder qu'à la methode.

35. Pour print horizontalement : system.out.PRINT()

Partie III

1/Listes – ArrayList

Importer :
java.util.ArrayList;

36. Créer une nouvelle liste : ArrayList<Type> nomDeLaListe = new ArrayList<Type>() ou Type est le type de valeur à insérer (string, int etc).

37. Le type d'une ArrayList doit comprendre une majuscule car Java reconnait 2 types de valeurs : primitives (int, double..) qui sont des types-valeurs comportant leurs propres valeurs. Et des types-réference qui comportent la reference de la place ou les valeurs de cette variable sont stockées. On peut y stocké un nombre infini de valeurs.

38. Ajouter des valeurs à une Liste : nomDeList.add(xxxx).

39. Recuperer une valeur d'une liste : nomDeListe.get(indexDePosition).

40. Connaitre le nombre de valeurs dans une liste : nomDeList.size()

41. Quand on ne connait pas la position d'une valeur dans une liste. :
nomDeListe.indexOf(nomDeVariable).
Et nomDeListe.lastIndexOf(nomDeVariable) pour connaitre la dernière occurrence de la valeur.
Exemple: quand on a deux meme valeurs dans une liste, on peut utiliser indexOf et lastIndexOf.

42. For-each loop est une variante de la loop for, son format est :
for (TypeDeVariable nomDeVariable : nomDeLaListe)
ou le TypeDeVariable est le type d'éléments de la liste et le nomDeVariable est la variable ou les valeurs vont etre stockées. Toute la liste sera faite, contrairement à une loop for ou on peut choisir le nombre d'iteration
Exemple:

```
ArrayList<String> teachers = new ArrayList<>();
teachers.add("Simon");
teachers.add("Samuel");
for (String teacher: teachers)
    System.out.println(teacher);
```

Son équivalent en une loop for :

```
ArrayList<String> teachers = new ArrayList<>();
teachers.add("Simon");
teachers.add("Samuel");
for (int i = 0; i < teachers.size(); i++) {
 String teacher = teachers.get(i)
// contents of  the for each loop;
System.out.println(teacher)
```

43. Methode remove pour retirer un element à un index donné : list.remove(1). On peut directement mettre en parametre ce qu'on veut remove si le parametre est de meme type que la liste. Exemple : une liste String peut accepter une methode list.remove("Cacapipi"). Mais une liste Integer ne peut pas accepter une methode list.remove(1), car l'int ici est l'index. Il faut convertir pour le faire :
list.remove(Integer.valueOf(1534546)

44. Methode contains pour checker si une valeur est présente dans une liste :
nomDeLaListe.contains(xxxxx). Elle retourne un boolean. If
(nomDeLaListe.contains("cacapipi") {
Sout("cacapipi is found!").

45. Une liste peut etre passée comme paramètre dans une methode.
Exemple : public static void print (ArrayList<String> list{
for (String valeur : list) {

System.out.println(valeur).

Elle printera toutes les strings de la liste.
On l'utilise ensuite quand on a une liste.
Exemple : ArrayList<String> strings = new ArrayList<>();
strings.add(xxx);
print(strings);
La variable passée en parametre de la methode n'a pas besoin d'étre la meme que celle initialement initialisée.

46. Une methode qui retourne une valeur n'a pas 'void' mais le type de valeur qu'elle renvoie à la place et utilise le ccommande 'return'.
Exemple : public static int (ArrayList<Integer> list){
return list.size();
On peut passer une liste comme paramètre dans une methode.
Exemple : removeLastNumber(listOfStrings)

47. NullPointerException : aucune instance deisponible

2/Tableaux – Arrays

48. Création (+ instanciation) d'une array à un size précis : typeD'Element[]
nomDeVariable = new typeD'Element[taille]
Instanciation : nomDeVariable= new typeD'Element[taille]
int[] nombre = new int[3] (elle acceptera 3 nombres)
String[] phrase = new String[5] (elle acceptera 5 phrases)

49. L'assignement d'une valeur à un index se fait manuellement. Ou de manière simplifiée : int[] nombre = {1, 2, 3, 4}
Exemple :
nombre[0] = 2
Pour y accéder : SOUT (nombre[0])

Deuxieme manière intermediaire :
type[] tab;
tab = new type[]{valeurs}

50. L'échange de 2 valeurs à leurs 2 index respectivement :
int echange = nombre[0];
nombre[0] = nombre[1];
nombre[1] = echange;

51. Pour parcourir toute une array, on n'utilise pas list.size() mais array.length dans une boucle while.
Exemple : int index = 0
while (index < array.length){
index++;

52. Array.length print le chiffre qui se trouve entre les crochets [] mais son size est length – 1. Ainsi, array[10] aura une array.length = 10, et un size = 9.

53. Pour éviter la repétition d'une réponse dans une boucle, on définit une variable en dehors de la loop. Notre réponse sera printed quand la variable atteindra la valeur atteinte, au lieu d'etre printed à chaque boucle.

54. Possibilité d'instancier une case selon 3 manières :
    Exemple :
    Point p = (1,2)
    tab[0] = p
    tab[1] = new Point (3,4)
    tab[2] = tab[0]

55. for (type valeur : array) {
    System.out.println(valeur).
56. Pour trouver une valeur dans un tableau :
    Exemple :
    int[] tab = {valeurs}
    System.out.println(Arrays.binarySearch(tab, 1)

3/Utiliser les Strings

57. Négation en utilisant '!'. Ex: if (!(text.equals("cake")) équivaut à : if text is not equal to cake

58. La méthode 'split' coupe les strings d'une array à l'endroit défini.
    NomDeVariable.split("endroit ou on veut que la phrase soit coupée")
    Exemple : String text = "un deux trois"
    String[] pieces = text.split(" ") ---- ici l'endroit à couper est les spaces. Donc à chaque space, la phrase sera coupée.
    SOUT(pieces[0] + pieces[1]) donne :
    un
    deux

59. Récupérer un caractère à un index spécifique : char nomDeVariable = nomDeLaString.charAt(numeroIndex)

JAVA – UPMC

Partie 1& 2 à récuperer
Partie III

1/Garbage collector

60. Destruction implicite : quand un object n'est plus utilisé, JAVA le détruit solo

61. Il existe 2 types de déstruction :

Dé-réferencement EXPLICITE (usage de =) :
Exemple :
Point p = new Point(1,3)
Point p2 = p;
p2 = new Point (2,4)

Dé-referencement IMPLICITE : création des variables dans un bloc dans le main
Exemple :
{
Point p = new Point(1,3)
System.out.println(p);
}

62. Si déclation de la variable est hors un bloc, il n'y pas de souci pour y accéder

    Exemple:
    Point p;
    {
    Point p = new Point(1,3)
    System.out.println(p);
    }
    System.out.println(p);

63. destruction d'une variable (quand elle est déclarée dans un bloc) =/= destruction d'une instance (quand elle n'est plus référencée)
64. Appel explicite du garbage collector pour lui dire de libérer IMMEDIATEMENT la mémoire (cas rares)
    System.gc()

2/Tableaux – Arrays

65. 2D
    int[][] matrice;
    matrice = new int [1][3]
    matrice[0][0] = 0;
    matrice[0][1] = 1;
    etc

Syntaxe alternative :

```
int nombre[3][4] = {
    {10,20,30,40},          //valeurs de la première rangée
    {15,25,35,45},          //valeurs de la deuxième rangée
    {1,2,3,4}               // valeurs de la troisième rangée
```

};

66. Accès : matrice.length // nombre de lignes
matrice[0].length // nombre de colonnes en se placant dans la premiere ligne

67. Manipulation des lignes :
Type[] ligne = variable[0]
System.out.println(ligne[0]) //valeurs de la premiere case
ligne[1] = nex Point(6,6) // nouvelles valeurs de la 2eme case

68. Matrice triangulaire :
Type[][] array = new Type[3][]
for (int I = 0; I < array.length; i++)
array[i] = new Type[i+1]

Output :
[]
[][]
[][][]

69. Seule egalite possible entre tableaux : comparaison d'entier et non pas de tableaux entiers

70. Comparaison des elements des tableaux :
Utilisation des return, break, continue
Exemple :
////////////////////////////////////

71.

Partie IV - Static

72. Informations qui pré-existent à la création d'une instance.
73. Une variable static est une zone commune indépendante des instances
74. La zone static ne connait pas les instances et est détruite à la fin du programme.

```
1  public class Point{
2    private static int cpt = 0; // initialisation obligatoire ici
3    private int id;  // initialisation interdite ici (-> constr)
4    private double x,y;
5
6    public Point(double x, double y){
7          this.x = x; this.y = y;
8          id = cpt++; // ou: id = cpt; cpt++;
9    }
10
11   // garantie de bonne gestion des id
12   public Point(){
13     this(Math.random()*10, Math.random()*10);
14   }
```

- Piège: attention aux constructeurs multiples
  ⇒ usage de `this()` très fortement conseillé pour passer toujours par le constructeur de référence et bien compter.

75. This en

argument veut dire que l'objet s'appelle soi-meme. Exemple d'un point qui s'ajoute soi-meme dans un tableau :
public Point (int x, int y) {
tab.add(this);

76. On déclaire une méthode static quand on veut retourner quelque chose sans new, juste en l'appelant :
public class Alea{
public static char lettre(){
return (char) (((char)(Math.random()…)

77. Invocation dans un main :
char c = Alea.lettre();

78. Pour éviter que les gens appeler une new Alea, on met son constructeur en private :
private Alea(); (code vide)

79. Il existe un constructeur sans argument automatique dans chaque fonction et c'est pour ça qu'on met le constructeur d'une méthode static en private
80. Une variable static prend une méthode static
81. Cas très particuliers, assez rare.

Partie V – Javadoc, débugging

82. Compilateur regarde la syntaxe, les types de variables et le niveau d'accès
83. JVM vérifie les instances
84. NullPointerException c'est quand on initialise pas
85. ArrayIndexOutOfBound nous donne l'index qui pose problème
86. Javadoc pour quand on connait l'objet à faire mais qu'on sait pas l'utiliser
87. Le type de commentaire : /**
…….
*/ génère du HTML
88. Je dois spécifier chaque argument et à quoi il sert en utilisant @param x
89.

Partie VI – Fiabilité et exceptions

90. JAMAIS DE SCROLL HORIZONTAL

91. Pour sécuriser certaines valeurs/variables, on la met en final : c'est l'équivalent d'une constante

92. Les variables finals sont écrites en majuscules

public final double x,l;

93. Interessant dans le cas ou on doit déclarer des variables publics mais qu'on veut pas qu'on veut pas qu'on y touche.
Exemple:
public final double x,y:
public Point(double x, double y){
this.x = y; this.y = y;}

94. Exploitation du NaN (main) : Double.isNan(nomDeVariable);

95. Utilisation du Nan (class): return Double.Nan

96. C'est un object : il se crée, se déclenche, et se récupère
Exemple :
RuntimeException e = new RuntimeException("Division par 0"); Déclaration
throw e; Déclenchement

97. Declaration + declenchement : throw new RuntimeException("Divison par 0");
98. Le throw plante le programme
99. On déclare l'exception à l'interieur de nos fonctions
Exemple :
public int depiler {
if (n <= 0){
throw new RuntimeException("impossible de retirer depuis 0");
100. Structure try & catch dans le main pour récuperer les exceptions :
try { code }
catch (<type exception><variable>){ code }

101. Catch déplante le programme contrairement à une seule déclaration simple de type throw
102. Les variables locales de try ne pourront pas etre accédées par le catch.

Partie VIII – Heritage

1/Overview

103. Récuperation d'un code existant sans faire de copie-coller

104. Syntaxe :
classeFille extends classeMère {

NE PAS DUPLIQUER LES ATTRIBUTS INITIALISES DANS LA CLASSE MERE

private String name;
Recuperation et declaration du constructeur
public Point(double x, double y, String name){
Obligation d'apeller super en premier lieu
super(x,y) //ce sont les attributs deja initialisés dans la classe mère
this.name = name;
Declaration de la méthode à ajouter
public String toString(){
code

105. Pour recuperer une méthode de la classe mere, on la precede par super.
Super.methodeDeClasseMere

106. La classe fille n'hérite pas des attributs et lmethodes privées. Ni des constructeurs public, protégés ou publics soient-ils.

107. Mots clés et leurs niveau d'accès :
public : accès direct par tout le monde
protected : accès par la classe fille et nul part ailleurs
private : accès que par la classe

108. Pour accèder à une variable protected dans la classe fille :
soit on utilise super.
Ou on initialise la variable direct :
int toto = id VS super.id
Mais vaut mieux utiliser super.id car il n'y a pas d'ambiguite et on sait d'ou provient le terme id.

2/Principe de subsomption

109. Polymorphisme de sorte à ce qu'on peut attribuer à la classe mère une méthode de la classe fille
PointMere p = new PointFille(1,2,"pipi");
110. Lors de l'affectation d'une classe fille à une classe mère, le compiler ne regarde que les variables de la classe mere. Donc appeler une méthode de la classe fille ne fonctionnera pas. C'est con ???
111. Toutes les classes dérivent de la classe Object (classe fantome) et donc héritent de tous ses attributs. Dans la classe Object on y trouve les méthodes de base comme clone(), toString(), equals() etc.
112. Lorsqu'on a 2 memes méthodes (une présente dans la classe mère et l'autre dans la fille), la JVM choisira la méthode la plus proche : soit celle de la classe fille
113. On peut forcer un programme à utiliser la méthode la classe mere en utilisant : super.nomDeMethode()
114. super =/= super()

- Par défaut, `equals` existe mais teste l'égalité référentielle, ce qui n'est pas intéressant...
- Redéfinition = faire en sorte de tester les attributs
  Un processus en plusieurs étapes:
  1. Vérifier s'il y a égalité référentielle:
     - si `true` renvoyer `true`
  2. Vérifier le type de l'`Object o` (cf prochain cours)
  3. Convertir l'`Object o` dans le type de la classe (idem)
  4. Vérifier l'égalité entre attributs

```java
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null) return false;
    if (getClass() != obj.getClass()) return false;
    Point other = (Point) obj;
    if (x != other.x) return false;
    if (y != other.y) return false;
    return true;
}
```

15/17

Partie IX – Conversion / Casting

115. dynamique : lors de l'execution du programme
116. On peut récuper des informations avec 'instanceof'. Mais normalement on en pas besoin si on a bien codé.
117.

Code spécifique dans les classes:

- dans Figure :

```java
public String getTypeFigure() {};
```

- dans Point :

```java
public String getTypeFigure() { return "Point"; }
```

- dans Rectangle :

```java
public String getTypeFigure() { return "Rectangle"; }
```

- Code générique (éventuellement en dehors des classes):

```java
public static void afficheType(Figure f) {
  System.out.println("C'est un "+f.getTypeFigure());
}
```

118. f.getTypeFigure() ira récuperer le type de figure de la classe appropriée.
Exemple :
Rectangle t = new Rectangle();
t.getTypeFigure() ira récuperer dans la classe Rectangle

119. getClass() retourne la classe complete de l'instance. Contrairement à instanceof qui repondait avec true et false.

120. Une conversion des objets n'implique aucune transformation. On ne peut convertir que le type de la variable qui réference.

121. Le probleme c'est que le compiler ne verifiera pas. On le force à faire une conversion qu'il veut pas.

Un système peu sécurisé à la compilation:

```
1 Point p1 = new Point("toto", 0, 2);
2 Point p2 = new Point( "toto 2", 3, 2);
3 Figure f = new Segment(p1, p2);
4
5 Figure f2 = (Figure) p1; // OK mais inutile
6 Figure f3 = p1; // OK Subsomption classique
7 Segment s = (Segment) f; // compilation OK
8 Point p3 = (Point) f; // compilation OK (!)
```

Exécution:

Crash du programme avec le message suivant

```
1 Exception in thread "main" java.lang.ClassCastException
2 cours2.Segment cannot be cast to cours2.Point
3         at cours2.Test.main(Test.java:26)
```

10,

122.

Cast: avec plusieurs niveaux de hiérarchie

```
1 // subsomption
2 Figure f = new Segment(p1, p2);
3 Figure f2 = new Carre(p1, cote);
4 Figure f3 = new Triangle(p1, p2, p3);;
5 Polygone p = new Triangle(p4, p5, p6);
6
7 // cast OK
8 Triangle t = (Triangle) p; // OK (comme precedemment)
9
10 Polygone p2 = (Polygone) f2; // OK compil + exec:
11                               // un Carre EST UN Polygone
12
13 Polygone p3 = (Triangle) f3; // OK:
14                     // f3 est un Triangle => conversion OK (JVM)
15                     // p3 peut referencer un Triangle OK (compil.)
16
```

123. Pour savoir si ma conversion d'objet est correcte, je me demande est que objet1 EST UN objet2 ? Si oui, je fais mon cast

124.

```
17 // cast KO
18 Carre c = (Carre) f;    // KO JVM
19 Cercle c = (Cercle) p; // KO Compil: opération impossible
```

Cast: sécurisation

Idée

Vérifier le type de l'instance avant la conversion

```
1        Figure f = new Segment(p1, p2);
2        Segment s;
3        if(f instanceof Segment)
4            s = (Segment) f;
```

⇒ vous utiliserez systématiquement cette sécurisation

```
5            return false;
6        // if (getClass() != obj.getClass()) en V1
7        if (!(obj instanceof Point))
8            return false;
9        Point other = (Point) obj;
10       if (x != other.x)
11           return false;
12       if (y != other.y)
13           return false;
14       return true;
15   }
```

Quel est le défaut de l'implémentation v2?

125. utiliser instanceof au lieu de getClass() est mauvais car instanceof test l'hierachie : une classe mère n'est pas égale à une classe fille.