

R1.01 : Initiation au développement

Cours 1

M.Adam, JF.Kamp, L.Naert

IUT de Vannes

3 août 2022

- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java
- 4 Premiers éléments du langage de Java
- 5 L'alternative
- 6 Les tests d'exécution
- 7 Conclusion

Le contenu de la séance

- Notion de programme
- Structure d'un programme
- Variables
- Affectation
- Séquence
- Alternative
- Entrées/Sorties

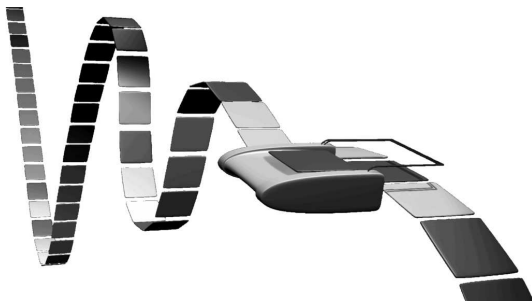
- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java
- 4 Premiers éléments du langage de Java
- 5 L'alternative
- 6 Les tests d'exécution
- 7 Conclusion

Pour wikipedia :

Un programme est une suite d'instructions qui spécifient étape par étape les opérations à exécuter par un ordinateur.

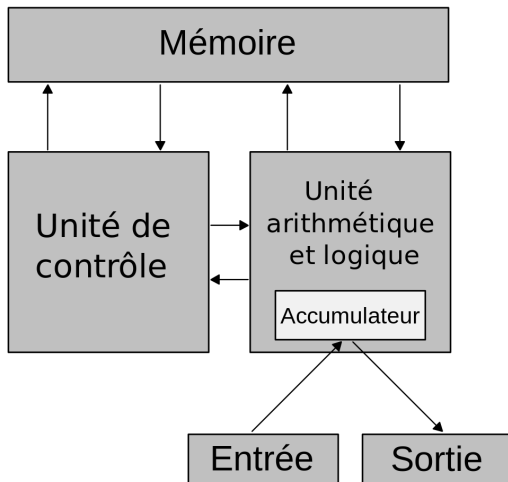
http://fr.wikipedia.org/wiki/Programme_informatique

Machine de Turing



- Un "ruban" infini divisé en cases consécutives.
- Une "tête de lecture/écriture".
- Un "registre d'état" qui mémorise l'état courant de la machine de Turing.
- Une "table d'actions"

Architecture Von Neumann I



Architecture Von Neumann II

- L'unité arithmétique et logique (UAL ou ALU en anglais) ou unité de traitement.
- L'unité de contrôle, chargée du séquençage des opérations ;
- La mémoire qui contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire volatile (programmes et données en cours de fonctionnement) et mémoire permanente (programmes et données de base de la machine).
- Les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.

Pour concevoir un programme ayant la puissance de calcul d'une machine de Turing, il faut et il suffit :

- des variables,
- la séquence,
- l'affectation,
- l'alternative,
- la boucle.

Tous les langages dit "impératifs" contiennent ces notions. Les autres notions sont là pour simplifier la vie du programmeur.

Exécution d'un programme

L'exécution d'un programme passe par :

- un compilateur
- un interpréteur

Compilation

Un programme qui traduit le texte (code source) dans un langage qui permettra son exécution, tel le langage machine, le bytecode ou le langage assembleur.



Première exemple

Le fichier source : calcul.c

```
#include <stdio.h>
#define TAUX 6.55957

int main () {
    float francs;

    francs=0;
    while (francs<=10) {
        printf("%4.1f francs = %.2f euros\n",francs,francs/TAUX);
        francs=francs+0.5;
    }

    return 0;
}
```

La compilation :

```
> cc calcul.c
> ls -l
total 20
-rwxrwxr-x 1 adam adam 7162 août  8 10:55 a.out
-rw-rw-r-- 1 adam adam  221 août  8 10:54 calcul.c
```

L'exécution :

```
> ./a.out
0.0 francs = 0.00 euros
0.5 francs = 0.08 euros
1.0 francs = 0.15 euros
...
9.5 francs = 1.45 euros
10.0 francs = 1.52 euros
```

Deuxième exemple

Le fichier source : Chaine.java

```
public class Chaine{

    private String name;

    public Chaine(String name){
        this.name = name;
    }

    public String getName(){
        return(name);
    }

    public void setName (String name){
        this.name = name;
    }

    public static void main(String[] args){
        Chaine ch;
        ch = new Chaine("Bonjour !");
        System.out.println(ch.getName());
        ch.setName("Au revoir !");
        System.out.println(ch.getName());
    }
}
```

La compilation :

```
> javac Chaine.java
```

```
ls -l
```

```
total 20
```

```
-rw-rw-r-- 1 adam adam 695 août 8 10:39 Chaine.class
```

```
-rw-rw-r-- 1 adam adam 423 août 8 10:39 Chaine.java
```

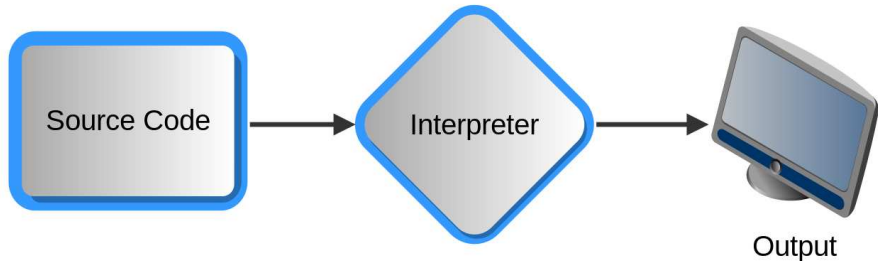
L'exécution :

```
> java Chaine
```

```
Bonjour !
```

```
Au revoir !
```


Un programme qui exécute les instructions demandées. Il joue le même rôle qu'une machine qui reconnaîtrait ce langage.



```
> python
Python 2.7.3 (default, Apr 10 2013, 05:46:21)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x=13
>>> resultat=x+2
>>> resultat=resultat*2
>>> print "Quand x vaut",x, ", le résultat vaut",resultat," !"
Quand x vaut 13 , le résultat vaut 30 !
>>>
```

Tout n'est pas aussi simple :

- un langage compilé peut aussi être interprété
- la compilation est syntaxique ou/et lexicale
- il existe des compilateurs de langages interprétés

- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java**
- 4 Premiers éléments du langage de Java
- 5 L'alternative
- 6 Les tests d'exécution
- 7 Conclusion

- Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld
- Dans le cadre de la ressource de R1.01, les aspects objets du langage ne seront pas abordés.
- L'éditeur de texte Geany sera utilisé (<https://www.geany.org/>)

- AlgoTouch :
 - <https://algotouch.irisa.fr/>,
 - <https://algotouch.irisa.fr/agt/>,
- JavaTutor : <http://www.pythontutor.com/java.html>,
- Jshell :
<https://docs.oracle.com/javase/9/jshell/introduction-jshell>

Structure du programme Java R1.01

```
/**
 * ROLE (Que fait-il ?)
 * blabla ...
 * @author
 **/
class MonAlgo {

    // point d'entrée du programme
    void principal() {
        // Déclaration des variables
        // instructions
    }
}
```

Le nom du fichier est MonAlgo.java.

Exemple programme java

```
/**
 * Calcul de la moyenne en de l'UE 1.1 du BUT Informatique/
 * @author M.Adam
 */
class CalculMoyenne {
    void principal() {
        double moyenneS101;
        double moyenneR101;
        double moyenneR102;
        double moyenneR110;
        double moyenneUE11;

        moyenneS101 = SimpleInput.getDouble("S1.01 Implémentation d'un besoin client : ");
        moyenneR101 = SimpleInput.getDouble("R1.01 Initiation au développement      : ");
        moyenneR102 = SimpleInput.getDouble("R1.02 Développement d'interfaces web   : ");
        moyenneR110 = SimpleInput.getDouble("R1.10 Anglais                          : ");

        moyenneUE11 = (moyenneS101 * 40
            + moyenneR101 * 42 + moyenneR102 * 12 + moyenneR110 * 6) / 100;
        System.out.println ("Votre moyenne de l'UE 1.1 : " + moyenneUE11);

        if (moyenneUE11 >= 10) {
            System.out.println ("Bravo, vous avez la moyenne en UE 1.1");
        } else {
            System.out.println ("Hélas, vous n'avez pas la moyenne en UE 1.1");
            if (moyenneUE11 >= 8) {
                System.out.println ("Mais vous avez plus de 8 en UE 1.1");
            } else {
                System.out.println ("Vous avez moins de 8, en UE 1.1");
            }
        }
    }
}
```


Sous PythonTutor et AlgoTouch

- PythonTutor <https://tinyurl.com/C01JTP01>,
- AlgoTouch <https://tinyurl.com/CC01AGTP01>.

Exécution du programme

En deux étapes :

- ❶ La **compilation** qui vérifie la "syntaxe" du programme.
- ❷ L'**exécution** qui déroule les instructions du programme.

Évidemment un programme qui n'est pas syntaxiquement correct ne peut être exécuté.

- La mise en œuvre de la compilation et de l'exécution avec Geany sera précisée en TP.
- Il est possible également de compiler et d'exécuter en ligne de commandes.

Sommaire

- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java
- 4 Premiers éléments du langage de Java**
- 5 L'alternative
- 6 Les tests d'exécution
- 7 Conclusion

- Les variables permettent de stocker des valeurs et des résultats de calcul.
- Une variable est identifiée par un nom et possède un type. Par convention, le nom d'une variable en java commence toujours par une lettre en minuscule et un mot interne doit commencer par une majuscule, par exemple `maVariable`.
- Une type permet d'identifier l'ensemble des valeurs susceptibles d'être stockées dans la variable. À rapprocher de $x \in Z$.

Java gère 8 types dits primitifs :

		Exemples	min	max
byte	1 octet	-1, 0, 30	-128	127
short	2 octets	-1, 0, 30	-32 768	32 767
int	4 octets	-1, 0, 30	-2^{31}	$2^{31} - 1$
long	8 octets	-1, 0, 30	-2^{63}	$2^{63} - 1$
float	4 octets	-1.4F, 0.0F, 30.56F	-1.40239846E-45	3.40282347E3863
double	8 octets	-1.4, 0.0, 30.56		
char	2 octets	'a', 'b', '='		
boolean	1 bit	true, false	false	true

Variables JavaTutor et AlgoTouch

- Variables JavaTutor

main:11	
moyenneS101	10.0
moyenneR101	10.0
moyenneR102	10.0
moyenneR110	10.0
moyenneUE11	10.0

- Variables AlgoTouch

mS101	mR101
7	7

Opérateurs

	Opérations	Comparaisons
byte	+ - * /	== != < <= > >=
short	+ - * /	== != < <= > >=
int	+ - * /	== != < <= > >=
long	+ - * /	== != < <= > >=
float	+ - * /	== != < <= > >=
long char	+ - * /	== != < <= > >=
boolean	&& !	== !=

La liste n'est pas exhaustive !

Négation

a	!a
true	false
false	true

Disjonction et conjonction

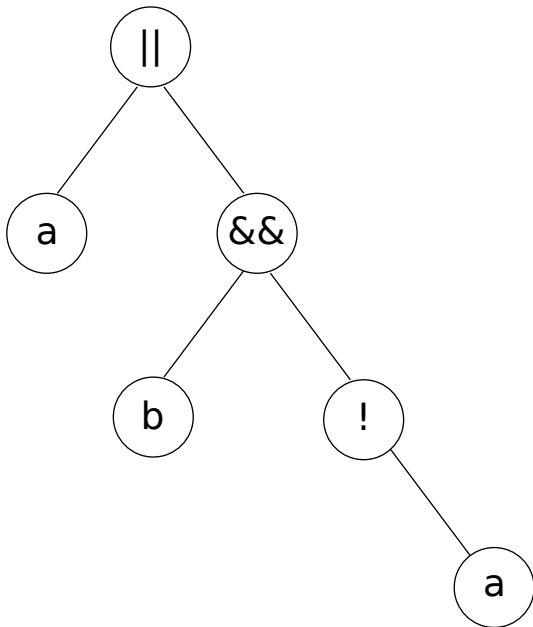
a	b	a b	a && b
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

- L'opérateur ! est **prioritaire** sur le || et &&.
- L'opérateur && est **prioritaire** sur le ||.
- Les opérateurs && et || ne sont pas **commutatifs en Java**.

L'évaluation

```
/**
 * Évaluation d'une expression logique
 * @author M.Adam
 */
class OuEtNon{
    void principal(){
        boolean a;
        boolean b;
        boolean c;

        a = false;
        b = true;
        c = a || b && !a;
        System.out.println ("c = " + c);
    }
}
```



Et si a et b sont faux ?

En cas de doute, parenthésez vos expressions :

`a + (b * c)`

`(a + b) * c`

`a || (b && c)`

`(a || b) && c`

```
"type" nomVar1 [ = valeur initiale ];  
final "type" NOM_CONSTANTE1 = valeur initiale;  
  
double moyenneS101;  
double moyenneR101;  
double moyenneR102;  
double moyenneR110;  
double moyenneUE11;  
  
final int C_S101 = 42;  
final int C_R101 = 40;  
final int C_R102 = 12;  
final int C_R110 = 6;
```

Une variable `final` est une constante et ne peut donc changer de contenu.

L'affectation

Le signe de l'affectation est `=`.

L'affectation permet de donner un contenu à une variable.

```
nbNotes = 3;  
somme = note1 + note2;  
somme = somme + 10;
```

- En AlgoTouch <https://tinyurl.com/C01AGTAff01>
- En Java Tutor <https://tinyurl.com/C01JTAff01>

- Il est possible de convertir :
byte → short → int → long → float → double
- Il est possible d'utiliser les opérateurs avec des types différents. Dans ce cas, Java procède à un changement du type des opérandes afin qu'ils soient toutes de même type pour pouvoir calculer l'expression. Ce changement de type se fait de telle sorte qu'il n'y ait pas de perte de précision, c'est-à-dire suivant la hiérarchie suivante :
byte → short → int → long → float → double

```
int i = 1;
double d = 1;
double sDouble = i + d; // est correct
int sInt = i + d; // est incorrect
```


Le type Chaîne String

Le type String **n'est pas un type primitif** en Java.

```
String mot1;  
String mot2;  
mot1 = "moto";  
mot2 = "culture";  
System.out.println(mot1 + mot2);  
System.out.println("Longueurmot1 : " + mot1.length());
```

Dans le cadre de la roussource R1.01, nous n'effectuerons pas de comparaison de chaîne de caractères.

Une séquence est une suite d'instructions. Les instructions d'une séquence sont séparées par un point-virgule (;).

```
moyenneProg = SimpleInput.getDouble("Moyenne en programmation");  
moyenneOMGL = SimpleInput.getDouble("Moyenne en OMGL");  
moyenneASR  = SimpleInput.getDouble("Moyenne en ASR");
```

Les entrées-sorties permettent de communiquer avec l'utilisateur du programme.

- `System.out.print()` et `System.out.println()`
- Saisie d'un int : `int SimpleInput.getInt()`
- Saisie d'un float : `float SimpleInput.getFloat()`
- saisie d'un double : `double SimpleInput.getDouble()`
- Saisie d'un boolean : `boolean SimpleInput.getBoolean()`
- Saisie d'un char : `char SimpleInput.getChar()`
- Saisie d'un String : `String SimpleInput.getString()`

```
System.out.print ("Votre moyenne en informatique");  
System.out.println (" est : " + moyenneINFO);  
System.out.println ();
```

```
moyenneProg = SimpleInput.getDouble("Moyenne en programmation");
```

- La librairie `Simpleinput` est utilisée dans le cadre de la ressource pour la saisie des données. Ce n'est pas une librairie standard. D'autres manières d'effectuer une saisie vous seront présentées dans d'autres modules de programmation.
- Par souci de simplification au niveau des types, nous vous conseillons, dans le cadre de la ressource R1.01, d'utiliser des `double` au lieu des `float`.

- Un programme syntaxiquement correct n'est pas un programme correct.
- Un programme correct est un programme qui produit le résultat attendu.

Un programme non correct

```
/**
 * Calcul de la moyenne en de l'UE 1.1 du BUT Informatique/
 * @author M.Adam
 */
class CalculMoyenne {
    void principal() {
        double moyenneS101;
        double moyenneR101;
        double moyenneR102;
        double moyenneR110;
        double moyenneUE11;

        moyenneS101 = SimpleInput.getDouble("S1.01 Implémentation d'un besoin client : ");
        moyenneR101 = SimpleInput.getDouble("R1.01 Initiation au développement      : ");
        moyenneR102 = SimpleInput.getDouble("R1.02 Développement d'interfaces web   : ");
        moyenneR110 = SimpleInput.getDouble("R1.10 Anglais                          : ");

        moyenneUE11 = moyenneS101 * 40
            + moyenneR101 * 42 + moyenneR102 * 12 + moyenneR110 * 6;
        System.out.println ("Votre moyenne de l'UE 1.1 : " + moyenneUE11);

        if (moyenneUE11 >= 10) {
            System.out.println ("Bravo, vous avez la moyenne en UE 1.1");
        } else {
            System.out.println ("Hélas, vous n'avez pas la moyenne en UE 1.1");
            if (moyenneUE11 >= 8) {
                System.out.println ("Mais vous avez plus de 8 en UE 1.1");
            } else {
                System.out.println ("Vous avez moins de 8, en UE 1.1");
            }
        }
    }
}
```


Un programme syntaxiquement faux

```
/**
 * Calcul de la moyenne en de l'UE 1.1 du BUT Informatique/
 * @author M.Adam
 */
class CalculMoyenne {
    void principal() {
        double moyenneS101;
        double moyenneR101;
        double moyenneR102;
        double moyenneR110;
        double moyenneUE11;

        moyenneS101 = SimpleInput.getDouble("S1.01 Implémentation d'un besoin client : ");
        moyenneR101 = SimpleInput.getDouble("R1.01 Initiation au développement      : ");
        moyenneR102 = SimpleInput.getDouble("R1.02 Développement d'interfaces web   : ");
        moyenneR110 = SimpleInput.getDouble("R1.10 Anglais                          : ");

        moyenneUE11 = (moyenneS101 * 40
            + moyenneR101 * 42 + moyenneR102 * 12 + moyenneR110 * 6) / 100;
        System.out.println ("Votre moyenne de l'UE 1.1 : " + moyenneUE11);

        if (moyenneUE11 >= 10) {
            System.out.println ("Bravo, vous avez la moyenne en UE 1.1");
        } else {
            System.out.println ("Hélas, vous n'avez pas la moyenne en UE 1.1");
            if (moyenneUE11 >= 8) {
                System.out.println ("Mais vous avez plus de 8 en UE 1.1");
            } else {
                System.out.println ("Vous avez moins de 8, en UE 1.1");
            }
        }
    }
}
```

- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java
- 4 Premiers éléments du langage de Java
- 5 L'alternative**
- 6 Les tests d'exécution
- 7 Conclusion

Pourquoi une alternative ?

- Avoir des valeurs dans un bon domaine (**important pour la sécurité**),
- éviter des erreurs d'exécution,
- résoudre un problème.

```
double note = SimpleInput.getDouble ("Note entre 0 et 20");
```

Il faut vérifier que la note est bien dans le domaine, comprise entre 0 et 20.

```
my = somme/nbNotes;
```

Il faut vérifier que le nombre de notes est supérieur à zéro.

Remarque : dans le cas d'une division par zéro, le résultat est *Infinity*. L'exécution se poursuit sans produire d'erreur au moment de la division par zéro.

Dans le calcul d'un quotient familiale :

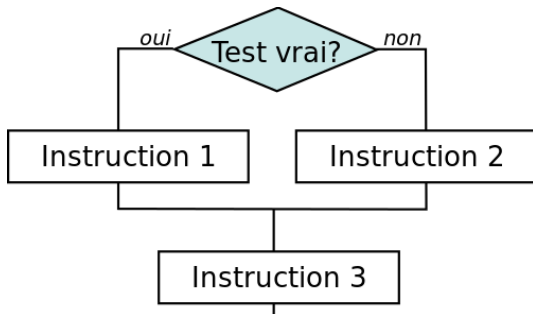
- un adulte compte pour une part,
- les deux premiers enfants comptent pour une demi-part,
- les enfants suivants comptent pour une part.

Il faut effectuer un calcul différenciant les enfants.

Alternative

```
if (condition booléenne){  
    instructions;  
[] else {  
    instructions;  
}
```

```
if (test) {  
    instruction 1;  
} else {  
    instruction 2;  
}  
instruction 3;
```



Alternative simple

```
if (nbNotes > 0) {  
    my = somme/nbNotes;  
}
```

Exemple complet

```
/**
 * Essai d'alternative simple
 * @author M.Adam
 */
class AlternativeSimple{

    void principal(){
        int    nbNotes;
        double somme;
        double my;

        somme = SimpleInput.getDouble ("Total des notes");
        nbNotes = SimpleInput.getInt ("Nombre de notes");
        if (nbNotes > 0) {
            my = somme/nbNotes;
            System.out.println ("Moyenne = " + my);
        }
    }
}
```

- En AlgoTouch <https://tinyurl.com/C01AGTAlt01>
- En Java Tutor <https://tinyurl.com/C01JTAlt01>

Alternative double

```
if (nbEnfants <= 2) {  
    qf = revenu / (nbAdultes + nbEnfants * 1/2);  
} else {  
    qf = revenu / (nbAdultes + 1 + nbEnfants - 2);  
}
```

Exemple complet

```
/*
 * Calcul du Quotient Familiale à partir du revenu, du nombre d'adultes
 * et du nombre d'enfants
 * @author M.Adam
 */
class QF{
    void principal(){
        int    revenu, nbAdultes, nbEnfants;
        double qf;

        revenu    = SimpleInput.getInt("Revenu en Euro");
        nbAdultes = SimpleInput.getInt("Nombre d'adultes");
        nbEnfants = SimpleInput.getInt("Nombre d'enfants");

        if (nbEnfants <= 2) {
            qf = revenu / (nbAdultes + nbEnfants * 1/2);
        } else {
            qf = revenu / (nbAdultes + 1 + nbEnfants - 2);
        }

        System.out.println("Quotient Familliale = " + qf);
    }
}
```

- En AlgoTouch : <https://tinyurl.com/C01AGTAlt02>
- En Java Tutor : <https://tinyurl.com/C01JTAlt02>

- Il est possible de réécrire certains codes en utilisant les propriétés de la logique.
- Deux programmes sont identiques s'ils produisent toujours la même exécution.

Les codes suivants :

if (a >= 8 && a < 10) {	if (a >= 8) {
a = a + 1;	if (a < 10) {
}	a = a + 1;
	}
	}

Exécutions identiques ou pas ?

Les codes suivants :

```
if (a < 5 || b < 5) {  
    a = a + 1;  
}
```

```
| if (a < 5) {  
|     a = a + 1;  
| }  
| if (b < 5) {  
|     a = a + 1;  
| }
```

Exécutions identiques ou pas ?

Les codes suivants :

if (a <= 19) {	if (a > 19) {
a = a + 1;	a = 20;
} else {	} else {
a = 20;	a = a + 1;
}	}

Exécutions identiques ou pas ?

Sommaire

- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java
- 4 Premiers éléments du langage de Java
- 5 L'alternative
- 6 Les tests d'exécution**
- 7 Conclusion

Un programme correct

Un code est correct s'il respecte les deux conditions suivantes :

- se termine normalement sans erreur,
- donne le résultat attendu.

Connaitre le résultat attendu

Il faut connaitre le résultat attendu par l'exécution du programme.

Exemple

Calcul de l'aire d'un cercle :

$$R \mapsto R$$

$$r \rightarrow r^2 \pi$$

$$10 \rightarrow 314,15$$

Que j'aime à faire connaitre ce nombre utile aux sages...

Tester tous les cas possibles

Il faut dans **tous les cas** que le programme donne un résultat correct. Le programme ne doit jamais rendre un résultat incorrect.

Exemple

```
/*
 * Calcul du Quotient Familiale à partir du revenu, du nombre d'adultes
 * et du nombre d'enfants
 * @author M.Adam
 */
class QF{
    void principal(){
        int    revenu, nbAdultes, nbEnfants;
        double qf;

        revenu    = SimpleInput.getInt("Revenu en Euro");
        nbAdultes = SimpleInput.getInt("Nombre d'adultes");
        nbEnfants = SimpleInput.getInt("Nombre d'enfants");

        if (nbEnfants <= 2) {
            qf = revenu / (nbAdultes + nbEnfants * 1/2);
        } else {
            qf = revenu / (nbAdultes + 1 + nbEnfants - 2);
        }

        System.out.println("Quotient Familliale = " + qf);
    }
}
```


Déterminer tous les cas à tester :

-
-
-
-
-

*"Le test de programmes permet de prouver la présence de bugs,
non leur absence."*

Edsger Dijkstra

- 1 Introduction
- 2 Notion de programme
- 3 Structure d'un programme Java
- 4 Premiers éléments du langage de Java
- 5 L'alternative
- 6 Les tests d'exécution
- 7 Conclusion

Les points à retenir

- Une programme est une suite d'instructions séparées par ;.
- Une variable permet de stocker des valeurs de types : int, float, char, boolean, String.
- L'affectation notée = permet de stocker une valeur dans une variable.
- l'alternative exprimée par if permet de modifier la séquentialité d'exécution des instructions.
- Les interactions avec l'utilisateur se font par `System.out.print()`, `System.out.println()`, `SimpleInput.getInt()`, etc.
- La compilation permet de vérifier, entre autre, la syntaxe du programme.
- L'exécution permet de vérifier les erreurs du programme. Un programme qui s'exécute en donnant le bon résultat n'est pas obligatoirement correct.

- Les boucles ou comment répéter plusieurs fois les mêmes instructions