

Algorithmique & Programmation

Vue d'ensemble et premiers pas

yann.secq@univ-lille.fr

ALMEIDA COCO Amadeu, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEPRETRE Éric, MARSHALL-BRETON Christopher, ROUZÉ Timothé, SECQ Yann, SOW Younoussa, SUE Yue

Objectifs du semestre

- Maîtriser les concepts fondamentaux de l'algorithmique et compétence de base en programmation
- Maîtriser les bases d'un langage impératif
- Déterminer les éléments pertinents d'un problème
- Savoir analyser et décomposer un problème et produire un programme y répondant
- **Réaliser deux projets (fermé et ouvert)**

INFORmation + autoMATIQUE

- Science du **traitement automatisé de l'information**
- Exprimer des traitements sous la forme d'**un ensemble d'actions élémentaires exécutables automatiquement** par une machine à calculer
- **Objectif du SI : apprendre à modéliser la résolution d'un problème avec un algorithme et produire un programme effectuant automatiquement cette résolution**
- Apprendre à observer, échanger, comprendre, décrire, organiser, vérifier, s'adapter aux changements ...

Concepts fondamentaux

- Langage, algorithme et programme (0)
- Type, variable et affectation (I)
- Condition, alternative (2) et répétition (3-4)
- Fonction (5-....)
- Assertion et test (5)
- Tableau (6)
- Création de type (9)
- Récursivité (II)

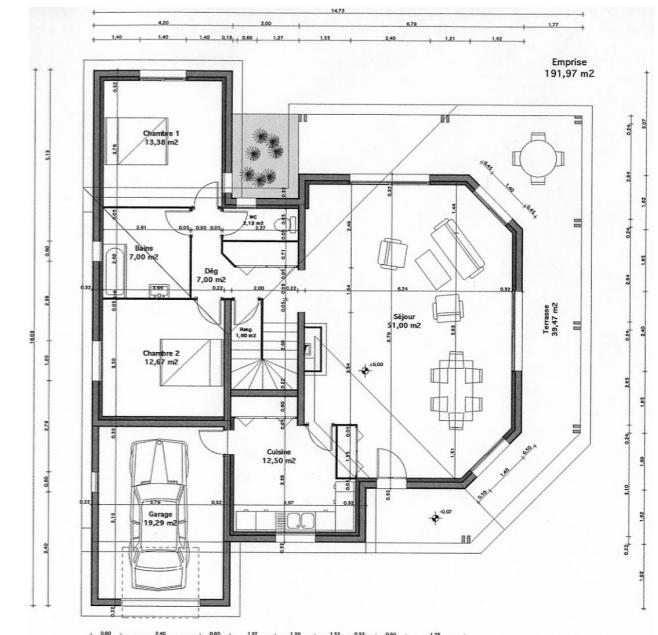
Concepts fondamentaux

- Langage, algorithme et programme
- Type, variable et affectation
- Condition, alternative et répétition
- Fonction
- Assertion et test
- Tableau
- Création de type
- Récursivité

Savoir analyser et décomposer un problème pour créer un programme y répondant

Langage, algorithme et programme

- Décrire un processus précisément nécessite **un langage formel**
- **Un algorithme est une description abstraite des étapes de résolution d'un (ensemble de) problème(s)**
- La description repose sur un petit nombre d'éléments de base pouvant être composés
- **Un programme est un algorithme exprimé dans un langage informatique**
- Un même algorithme peut être implémenté en différents programmes ...



Code source sauvegardé dans un fichier texte nommé Go.java

```
class Go extends Program {  
    void algorithm() {  
        print("Let's go !");  
    }  
}
```

1. Édition

On exécute le code machine avec l'interpréteur Java (commande ijava)

ijava Go

Avec ce programme, on obtient cet affichage

```
yannsecq@MBP exemples % ijavac Go.java  
yannsecq@MBP exemples % ijava Go  
Let's go !  
yannsecq@MBP exemples %
```

3. Exécution

On compile le code source en code machine à l'aide du compilateur (commande ijavac)

ijavac Go.java

On obtient le code machine (illisible pour un humain) dans le fichier Go.class

```
Êþº¾^@^@^@^6^@^W  
^@^E^@^P^H^@^Q  
^@^D^@^R^G^@^S^G^@^T^A^@^F<init>^A^@^C()V^A^@^DCode^A^@^0LineNum  
berTable^A^@^RLocalVariableTable^A^@^Dthis^A^@^GLGo;^A^@  
algorithm^A^@  
SourceFile^A^@  
Go.java^L^@^F^@^G^A^@^0Let's go !  
^L^@^U^@^V^A^@^EHello^A^@^GProgram^A^@^Eprint^A^@^U(Ljava/lang/  
String;)V@ ^@^D^@^E^@^@^@^@^B^@^@^@^F^@^G^@^A^@^H^@^@^@/  
^@^A^@^A^@^@^@^E* ^@^A±^@^@^@^B^@ ^@^@^@^F^@^A^@^@^@^A^@  
^@^@^@^L^@^A^@^@^@^E^@^K^@^L^@^@^@^@^@^M^@^G^@^A^@^H^@^@^@^5^@^B^@  
^A^@^@^@^@^G^@^R^B¶^@^C±^@^@^@^B^@ ^@^@^@^@^C^@^F^@^D^@  
^@^B^@^@^@^@^C^@^F^@^D^@ ^@^@^@^@^L^@^A^@^@^@^G^@^K^@^L^@^@^@^A^@^N^@^@^B^@^0
```

2. Compilation

Fonctionnement différent des langages interprétés, typiquement python ...

Compilation et exécution

- **Code source** (.java) : programme écrit dans un fichier texte
- **Code machine** (.class) : code exécutable par la machine
- **Compilation** (javac) : transformation du code source en code machine
- **Exécution** (java) : interprétation du code machine par la machine (virtuelle Java)
- **Bug** : démonstration de nos défaillances à anticiper certaines situations ;)

Traces d'activité en TP

- **An passé** : relevé manuel en fin de TP de l'avancement de chacun·e (vous situez, éléments pour organiser le tutorat)
- **Cette année** : expérimentation d'automatisation avec des traces lors de chaque compilation (`ijavac`) et exécution (`ijava`)
- **Trois objectifs principaux sont visés**
 - **Pédagogiques côté étudiant·e·s**, vous situer pour rapport à votre groupe ou la promotion,
 - **Pédagogiques côté enseignant·e·s** pour faciliter le suivi en TP et affiner l'allocation des tuteurs et tutrices (rapidement !)
 - **Scientifiques** : faire des études pour caractériser les erreurs les plus courantes et les situations posant le plus de difficultés (et comment les atténuer)

Concrètement ?

- Cours (1h30) / Pré-TD (1h30) / TD (1h30) et TP (3h)
- Même langage en cours / TD / TP
- *i*java = Java mais **sans objets/classes (S2)**
- Outils : un traitement de texte + Java (JDK)
- **Avoir votre propre ordinateur est important**

Algorithmique & Programmation

Type, variable & affectation

yann.secq@univ-lille.fr

ALMEIDA COCO Amadeu, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEPRETRE Éric, MARSHALL-BRETON Christopher, ROUZÉ Timothé, SECQ Yann, SOW Younoussa, SUE Yue

Objectifs du cours

- Notion de type
- Types de base
- Notion de variable
- Notion d'affectation
- Structuration d'un programme i java
- Trace d'exécution d'un programme

Notion de type

- **Informatique = Information + (traitement) automatique**
- Qu'est-ce qu'une **information** ?
- Différentes natures d'information:
 - Nombre, texte, image, son ...
 - Différents types d'opérations ...
- Une fois numérisée (binaire !), on parlera de **donnée**
- **Un type correspond à la nature d'une donnée**
- Pour l'instant : les nombres, les caractères et les chaînes de caractères

Types de base

TYPE	VALEURS	OPÉRATEURS/FONCTIONS
Nombre entier	-4, 0, 42	+, -, *, /, %, ==
Nombre réel	-5.23, 0.0, 3.14159	+, -, *, /, %, ==
Caractère	'a', '4', '\$', ''	+, -, ==
Chaînes de caractères	"Hello" , "z", " ", ""	charAt, substring, equals
Booléen	true, false	<i>Prochain cours ;)</i>
Tableau	des différents types ci-dessus	<i>Futur cours ;)</i>

Booléen

- Représente une information ne pouvant prendre que 2 états (vrai ou faux)
- Boolean : true ou false
- Opérations : cf prochain cours :)

Représenter des nombres

- **Nombres entiers**

- byte, short, int et long
- exemples: 0, -1, 32, -456, ...

- **Nombres décimaux**

- float et double
- exemples: -0.543, 123214.123123, ...

- ATTENTION : 0 et 0.0 sont de types différents

- **Opérateurs:** +, -, *, /, %, ==

Expression arithmétique

- Expression constituée de nombres et d'opérateurs
- Une expression arithmétique calcule un nombre lorsqu'elle est évaluée
- Exemple: $42 + (2 / 3) * -17 \Rightarrow ?$
- Abusez des parenthèses !

Expression	Evaluation
$3 + 5$	
$12 / (2+1)$	
$10 / 6$	
$10 \% 6$	

Représenter du texte



Аа Бб Вв Гг Дд Ее
Ёё Жж Зз Ии Йй
Кк Лл Мм Нн Оо
Пп Рр Сс Тт Уу Фф
Хх Цц Чч Шш Щщ
Ңъ Ыы Ъъ Ээ Юю
Яя (Ii Θθ Vv Ѓѣ)

夢	風	火	水	心	勇	美
Dream	Wind	Fire	Water	Heart	Courage	Beauty
雨	雪	宝	金	花	星	死
Rain	Snow	Treasure	Gold	Flower	Star	Death
猫	犬	日	月	愛	秋	夏
Cat	Dog	Sun	Moon	Love	Autumn	Summer
考	強	空	朝	夜	冬	春
Think	Strength	Sky	Morning	Night	Winter	Spring

Représenter du texte

- Notion d'alphabet (symbole) et de mots (ensemble de symboles)
- **Représenter un seul caractère : char**
 - `char` (code UTF-16, cf. cours de codage)
 - ex: '`a`', '`Y`', '`9`', '', '`õ`' ...
- **Représenter un ensemble de caractères : String**
 - `String` (chaîne de caractères)
 - ex: "`Hello`", "", "Hamlet: \"Words, words, words\""

Opérations sur les chaînes

Expression	Evaluation	Type du résultat
equals ("titi", "tutu")		
charAt ("tutu", 0)		
length ("Hello")		
substring ("Bonjour", 0, 3)		
"Hello" + "World"		

Opérations sur les chaînes

Expression	Evaluation	Type du résultat
equals ("titi", "tutu")		boolean
charAt ("tutu", 0)		char
length ("Hello")		int
substring ("Bonjour", 0, 3)		String
"Hello" + "World"		String

Opérations sur les chaînes

Expression	Evaluation	Type du résultat
equals ("titi", "tutu")	false	boolean
charAt ("tutu", 0)	't'	char
length ("Hello")	5	int
substring ("Bonjour", 0, 3)	"Bon"	String
"Hello" + "World"	"HelloWorld"	String

Opérations sur les caractères et les chaînes de caractères

DÉFINITION	USAGE		
Nom de la fonction et types des paramètres	Type du résultat	Fonctionnalité	Exemple
length(String)	int	Fonction retournant le nombre de caractères constituant la chaîne.	length("Hello") => 5
charAt(String, int)	char	Fonction retournant le caractère se trouvant à l'indice donnée dans la chaîne.	charAt("Hello", 0) => 'H'
substring(String, int, int)	String	Fonction copiant les caractères de la chaîne compris entre un indice de début (inclus) et un indice de fin (exclus).	substring("Hello", 0, 4) => "Hell"
equals(String, String)	boolean	Fonction vérifiant que les deux chaînes contiennent exactement les mêmes caractères.	equals("Hello", "Hello") => true equals("Hello", "hello") => false
String + String String + char String + int	String	Opérateur fabriquant une nouvelle chaîne à partir d'une chaîne et n'importe quelle autre valeur.	"Hel" + "lo" => "Hello" "Hell" + 'o' => « Hello » "Hell" + 0 => "Hello"

NE PAS CONFONDRE DÉFINITION ET USAGE D'UNE FONCTION
**Pour l'instant, le seul endroit où vous utiliserez des types,
c'est lorsque vous déclarerez des variables.**

Opérations d'entrées/sorties

DÉFINITION	USAGE		
Nom de la fonction et types des paramètres	Type du résultat	Fonctionnalité	Exemple
print(*)	void	Fonction affichant à l'écran les données	print("Hello") print(42)
println(*)	void	Fonction affichant à l'écran les données suivies d'une retour à la ligne	println("Hello") println(42)
readInt()	int	Fonction renvoyant un entier saisi par l'utilisateur au clavier.	int age = readInt()
readDouble()	double	Fonction renvoyant un réel saisi par l'utilisateur au clavier.	double taux = readDouble()
readChar()	char	Fonction renvoyant un caractère saisi par l'utilisateur au clavier.	char lettre = readChar()
readString()	String	Fonction renvoyant une chaîne de caractères saisi par l'utilisateur au clavier.	String nom = readString()

Les fonctions print/println peuvent prendre n'importe quelle valeur comme paramètre !

Variable et affectation

- **Type** = nature d'une information
- **Variable** = emplacement mémoire nommé pouvant contenir une valeur d'un type de base
- A sa création, une variable n'est pas initialisée
- Ex: int age; String nom; char symbole, direction;
- **L'opération d'affectation = permet d'initialiser ou modifier le contenu d'une variable**
 - <nom_de_la_variable> = <valeur>
 - <nom_de_la_variable> = <expression>

Exemples d'affectation

- L'**affectation** stocke une valeur dans une variable
- Syntaxe: <variable> = <valeur> ou <expression>;
- Ex : age = 7; symbole = charAt ("Nord", 0);
- Initialisation à la déclaration : int age = 7;

La partie droite est évaluée et le résultat est stocké dans la variable indiquée à gauche

Notion de constante

- Parfois, on souhaite définir une valeur ne variant pas !
- Une constante est une variable dont la valeur ne peut changer
- Exemple : final double TVA = **19.6**;
- Préfixer le type par **final** pour définir une constante
- Pour distinguer constantes et variables, les constantes sont définies en MAJUSCULES (convention)

Forçage de type

- Parfois, on souhaite convertir un type vers un autre
- On peut le faire entre types numériques, mais aussi avec les caractères (cf. code ASCII) !

```
int note = (int) 10.5;
double ration = (double) 32;
double moyenne = (double) ((note+note) / 2);
int codeAscii = (int) 'A';
codeAscii = codeAscii + 1; // B !
char lettre = (char) codeAscii;
```

La table de codage ASCII

ASCII	Caractère														
0	NUL	32	Space	64	@	96	`	128	€	160	ি	192	À	224	à
1	SOH	33	!	65	A	97	a	129	à	161	í	193	Á	225	á
2	STX	34	"	66	B	98	b	130	,	162	ç	194	Â	226	â
3	ETX	35	#	67	C	99	c	131	f	163	£	195	Ã	227	ã
4	EOT	36	\$	68	D	100	d	132	"	164	¤	196	Ä	228	ä
5	ENQ	37	%	69	E	101	e	133	...	165	¥	197	Å	229	å
6	ACK	38	&	70	F	102	f	134	†	166	-	198	Æ	230	æ
7	BEL	39	'	71	G	103	g	135	‡	167	§	199	Ç	231	ç
8	BS	40	(72	H	104	h	136	^	168	..	200	È	232	è
9	HT	41)	73	I	105	i	137	‰	169	©	201	É	233	é
10	LF	42	*	74	J	106	j	138	š	170	¤	202	Ê	234	ê
11	VT	43	+	75	K	107	k	139	‘	171	«	203	Ë	235	ë
12	FF	44	,	76	L	108	l	140	Œ	172	¬	204	Ì	236	ì
13	CR	45	-	77	M	109	m	141	ž	173	®	205	Í	237	í
14	SO	46	.	78	N	110	n	142	Ž	174	®	206	Î	238	î
15	SI	47	/	79	O	111	o	143	܂	175	-	207	Ï	239	ï
16	DLE	48	0	80	P	112	p	144	,	176	◦	208	Đ	240	đ
17	DC1	49	1	81	Q	113	q	145	,	177	±	209	Ñ	241	ñ
18	DC2	50	2	82	R	114	r	146	"	178	²	210	Ò	242	ò
19	DC3	51	3	83	S	115	s	147	"	179	³	211	Ó	243	ó
20	DC4	52	4	84	T	116	t	148	"	180	‘	212	Ô	244	ô
21	NAK	53	5	85	U	117	u	149	•	181	μ	213	Ӯ	245	ö
22	SYN	54	6	86	V	118	v	150	-	182	¶	214	Ö	246	ö
23	ETB	55	7	87	W	119	w	151	-	183	•	215	×	247	÷
24	CAN	56	8	88	X	120	x	152	-	184	,	216	Ø	248	ø
25	EM	57	9	89	Y	121	y	153	™	185	ı	217	Ù	249	ù
26	SUB	58	:	90	Z	122	z	154	š	186	ő	218	Ú	250	ú
27	ESC	59	;	91	[123	{	155	,	187	»	219	Û	251	û
28	FS	60	<	92	\	124		156	œ	188	½	220	Ü	252	ü
29	GS	61	=	93]	125	}	157	ž	189	¾	221	Ý	253	ý
30	RS	62	>	94	^	126	~	158	ÿ	190	¾	222	Þ	254	þ
31	US	63	?	95	-	127	DEL	159	ÿ	191	¿	223	ß	255	ÿ

Notion de programme

*Nom du programme
(le fichier doit être
nommé Go.java)*

*Définition de
l'algorithme
principal du
programme*

```
class Go extends Program {  
    void algorithm() {  
        print("Let's go !");  
    }  
}
```

*On indique ici que
l'on souhaite
définir un
programme iJava*

*Appel de la fonction
prédéfinie print qui
affiche à l'écran
l'information contenue
dans les parenthèses*

- Un programme est **un fichier texte**
- Un programme doit définir un algorithme
 - void algorithm() { ... }
- **ATTENTION :**
 - Nom du fichier = nom du programme + « .java »
 - Vérifier le parenthésage (ouvrante/fermante)
 - La casse est importante (minuscule != majuscule)

Exemple de programme ijava

```
Nom du programme  
(Chute) → class Chute extends Program {  
  
Définition de  
l'algorithme principal → void algorithm() {  
  
Définition de  
variables →     int hauteur = 10;  
                 final double G = 9.81; // m.s-2  
                 double vitesse = sqrt(2*G*hauteur);  
  
Définition d'une  
constante →         print(vitesse); // 14.007  
  
Instruction  
d'affichage →     } } }
```

Programme défini dans un fichier nommé *Chute.java*

Trace d'exécution

- **Programme** = suite d'instructions
- **Exécution** = évaluation séquentielle d'une suite d'instructions
- **Trace** = visualisation de l'évolution de l'état des variables lors de l'exécution du programme
- A terme, simulation d'exécution "de tête" :)

Trace du programme Chute

```
1 class Chute extends Program {  
2     void algorithm() {  
3         final double G = 9.81; // m.s-2  
4         int hauteur = 10;  
5         double vitesse = sqrt(2*G*hauteur);  
6         print(vitesse); // 14.007  
7     }  
8 }
```

Entrée(s)	G	hauteur	vitesse	Sortie(s)
3	-	9.81		
4	9.81	10		
5	9.81	10	14,007	
6	9.81	10	14,007	14,007

Si l'on souhaite traiter différentes hauteurs ?

- **Un programme traite généralement une famille de problème**
- Comment généraliser notre programme précédent ?
- Introduction de la notion d'**entrée**, ici sous la forme d'une **saisie** auprès de l'usager
- A part l'initialisation de la hauteur, le programme reste identique !

Programme i java

```
class ChuteAvecSaisie extends Program {  
    void algorithm() {  
        final double G = 9.81; // m.s-2  
        int hauteur;  
        double vitesse;  
  
        println("Quelle hauteur ?");  
        hauteur = readInt(); ←  
        vitesse = sqrt(2*G*hauteur);  
        print(vitesse); // 14.007  
    }  
}
```

Affichage afin que l'usager sache ce qui est attendu de sa part

Instruction réalisant une saisie au clavier et retournant la valeur saisie sous la forme d'un entier

```

1 class Chute extends Program {
2     void algorithm() {
3         final double G = 9.81; // m.s-2
4         int hauteur;
5         double vitesse
6         print("Quelle hauteur ?");
7         hauteur = readInt();
8         vitesse = sqrt(2*G*hauteur);
9         print(vitesse); // 14.007
10    }
11 }
```

Entrée(s)	G	hauteur	vitesse	Sortie(s)
3	-	9.81		
4		9.81		
5		9.81		
6		9.81		Quelle hauteur ?
7	10	9.81	10	
8		9.81	10	14.007
9		9.81	10	14.007

Synthèse

- Notion de type et opérations associées
- Notion de variable et d'affectation
- Notion de forçage de type
- Notion de programme
- Exemple d'un programme ijava
- Notion de trace d'exécution

