Langage et algorithmique

A. Malek TOUMI

ENSTA Bretagne

Sommaire

1. **Notion de variable**
   - Déclaration
   - Types
   - Portée
   - Opérations
2. **Instructions du langage Java**
   - Commentaires
   - Littéraux (constantes)
   - Mots clés
   - Branchements conditionnels
   - Boucles
3. **Premiers programmes**

Sommaire

1. **Notion de variable**
   - Déclaration
   - Types
   - Portée
   - Opérations
2. **Instructions du langage Java**
   - Commentaires
   - Littéraux (constantes)
   - Mots clés
   - Branchements conditionnels
   - Boucles
3. **Premiers programmes**

Généralités

- Variable : donnée de base manipulée par le programme
- Emplacement mémoire (adresse binaire) contenant des données (binaire)
- Données de taille variable
- Langage de haut niveau : cache la gestion de la mémoire

Variables en Java

- Nécessité de déclarer les variables
- Noms de variables sensibles à la casse
- Règles de nommage :
  - commence par une lettre, un “_” ou un “$”
  - peut contenir des chiffres
  - ne peut pas contenir de séparateur
  - ne doit pas être un mot clé
  - peut contenir des accents, mais déconseillé
- Choisir des noms parlants (rester raisonnable)

Variables en Java

- Nécessité de déclarer les variables
- Noms de variables sensibles à la casse
- Règles de nommage :
  - commence par une lettre, un "_" ou un "$"
  - peut contenir des chiffres
  - ne peut pas contenir de séparateur
  - ne doit pas être un mot clé
  - peut contenir des accents, mais déconseillé
- Choisir des noms parlants (rester raisonnable)

```
surelyReachableObjectsWhichHaveToBeMarkedAsSuch
```

Sommaire

1. **Notion de variable**
   - Déclaration
   - Types
   - Portée
   - Opérations
2. **Instructions du langage Java**
   - Commentaires
   - Littéraux (constantes)
   - Mots clés
   - Branchements conditionnels
   - Boucles
3. **Premiers programmes**

Typage des variables

- Java : langage fortement typé

$\Rightarrow$ typage lors de la déclaration

- Vérification de la compatibilité des types par le compilateur
- Types divisés en deux catégories :
  - types primitifs
  - types composites

Types primitifs

- Entiers signés (complément à deux)

Types primitifs

Type	Taille	valeur min	valeur max
byte	1	$-2^7 = -128$	$2^7 - 1 = 127$
short	2	$-2^{15} = -32768$	$2^{15} - 1 = 32767$
int	4	$-2^{31} \simeq -2.14E9$	$2^{31} - 1 \simeq 2.14E9$
long	8	$-2^{63} \simeq -9.22E18$	$2^{63} - 1 \simeq 9.22E18$

Types primitifs

- Entiers signés (complément à deux)
- Réels (IEEE 754)

Types primitifs

Type	Taille	Min	Max	ε
float	4	1.402E-45	3.402E38	1.19E-7
double	8	4.94E-324	1.79E308	2.22E-16

Types primitifs

- Entiers signés (complément à deux)
- Réels (IEEE 754)
- Booléens (**true** et **false**)

Types primitifs

- Entiers signés (complément à deux)
- Réels (IEEE 754)
- Booléens (**true** et **false**)
- Caractères (unicode 16 bits)

Déclarations

Exemple (Quelques déclarations)

```
int i;
int reponse = 42;
double pi = 3.1416;
boolean conditionValide = true;
char c='a';
```

Types composites

Deux types composites :

- Les classes
  - classes de l'API Java
  - classes définies par l'utilisateur

Types composites

Deux types composites :

- Les classes
  - classes de l'API Java
  - classes définies par l'utilisateur
- Les tableaux

Types composites

Deux types composites :

- Les classes
  - classes de l'API Java
  - classes définies par l'utilisateur
- Les tableaux

Exemple (Classes de l'API)

- classes numériques : Integer, Double, ...
- chaînes de caractères : String

Sommaire

1 **Notion de variable**
- Déclaration
- Types
- Portée
- Opérations

2 **Instructions du langage Java**
- Commentaires
- Littéraux (constantes)
- Mots clés
- Branchements conditionnels
- Boucles

3 **Premiers programmes**

Portée, bloc

Portée

- Portée : zone de visibilité d'une variable
- Notion fondamentale
- Utilise la notion de bloc

Portée, bloc

Portée

- Portée : zone de visibilité d'une variable
- Notion fondamentale
- Utilise la notion de bloc

Définition (Bloc)

Ensemble des instructions entre une paire balancée d'accolades (accolade ouvrante et accolade fermante correspondante).

Exemple de blocs

Exemple (Bloc)

```
public class MaClasse
{// debut bloc 1
     private int variable1;
     public void essai()
     {// debut bloc 2
          int i, j;
     }// fin bloc 2
     public static void main(String args[])
     {// debut bloc 3
          int i;
     }// fin bloc 3
}// fin bloc 1
```

Effet de bord

Attention

Limiter la portée des variables au maximum : attention aux effets de bord

Effet de bord

Attention

Limiter la portée des variables au maximum : attention aux effets de bord

Définition (Effet de bord)

Effet de bord vient de **side effect** (effet secondaire). Modification d'une variable globale par une fonction.

Effet de bord

Attention

Limiter la portée des variables au maximum : attention aux effets de bord

Définition (Effet de bord)

Effet de bord vient de **side effect** (effet secondaire). Modification d'une variable globale par une fonction.

Le comportement d'un programme change selon l'ordre d'appel des fonctions à effet de bord.

Transtypage / cast

- Objectif : convertir d'un type vers un autre.
- Exemple : `double f = (double) retourInt ();`
- Attention : risque de perte de données (ex : **long** vers **int**)

Transtypage / cast

- Objectif : convertir d'un type vers un autre.
- Exemple : `double f = (double) retourInt ();`
- Attention : risque de perte de données (ex : **long** vers **int**)
- Transtypage implicite

Transtypage / cast

Type initial	Transtypage
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

Transtypage / cast

- Objectif : convertir d'un type vers un autre.
- Exemple : `double f = (double) retourInt ();`
- Attention : risque de perte de données (ex : **long** vers **int**)
- Transtypage implicite
- Ex : `if ('a' < 3)` ...

Transtypage / cast

- Objectif : convertir d'un type vers un autre.
- Exemple : `double f = (double) retourInt ();`
- Attention : risque de perte de données (ex : **long** vers **int**)
- Transtypage implicite
- Ex : `if ('a' < 3)` ...
- 'a' est converti en entier

Sommaire

1 **Notion de variable**
- Déclaration
- Types
- Portée
- Opérations

2 **Instructions du langage Java**
- Commentaires
- Littéraux (constantes)
- Mots clés
- Branchements conditionnels
- Boucles

3 **Premiers programmes**

Affectation

Définition (Affectation)

Affectation : donner une valeur à une variable. Réalisée par l'opérateur "=".

Affectation

Définition (Affectation)

Affectation : donner une valeur à une variable. Réalisée par l'opérateur "=".

Exemple (Affectations)

```
int a = 2;
a = a+1;
boolean b=(a>2);
char c = 'a'+3;
```

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i += 1 (ou i = i+1)

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i += 1 (ou i = i+1)
- $i++;$

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i $+=$ 1 (ou i $=$ i$+$1)
- $i++;\ i--;$

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i $+=$ 1 (ou i $=$ i$+$1)
- $i++$; $i--$;
- $++i$;

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i += 1 (ou i = i+1)
- $i++$; $i--$;
- $++i$; $--i$;

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i $+=$ 1 (ou i $=$ i+1)
- $i++;\ i--;$
- $++i;\ --i;$

Exemple

```
int j, i=0;
j = i++;
```

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i $+=$ 1 (ou i $=$ i$+$1)
- $i++$; $i--$;
- $++i$; $--i$;

Exemple

```
int j, i=0;
j = i++;      (j vaut 0, i vaut 1)
```

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i $+=$ 1 (ou i $=$ i$+$1)
- $i++$; $i--$;
- $++i$; $--i$;

Exemple

```
int j, i=0;
j = i++;      (j vaut 0, i vaut 1)
j = ++i;
```

Pré et post incrémentation

Exemple (Forme condensée)

```
a += 2;
b *= 3;
```

- Affectation très utile : i $+=$ 1 (ou i $=$ i$+$1)
- $i++$; $i--$;
- $++i$; $--i$;

Exemple

```
int j, i=0;
j = i++;      (j vaut 0, i vaut 1)
j = ++i;      (j vaut 2, i vaut 2)
```

Comparaisons

Opérateur	Exemple	Signification
>	a > 10	strictement supérieur
<	a < 10	strictement inférieur
>=	a >= 10	supérieur ou égal
<=	a <= 10	inférieur ou égal
==	a == 10	égal à
!=	a != 10	différent de

Opérations logiques

Opérateur	Exemple	Signification
`!`	`!a`	NON logique
`&&`	`a && b`	ET logique : évaluation paresseuse
`||`	`a || b`	OU logique : évaluation paresseuse
`&`	`a & b`	ET logique
`^`	`a ^ b`	OU exclusif logique
`|`	`a | b`	OU logique

Opérations logiques

Opérateur	Exemple	Signification
`!`	`!a`	NON logique
`&&`	`a && b`	ET logique : évaluation paresseuse
`||`	`a || b`	OU logique : évaluation paresseuse
`&`	`a & b`	ET logique
`^`	`a ^ b`	OU exclusif logique
`|`	`a | b`	OU logique

Remarque : évaluation paresseuse

Opérations arithmétiques

- Opérations de base : `+ - * /`
- Modulo : `%`
- Attention à la division entière :
  - 1/2 vaut 0
  - 1/2.0 vaut 0.5

Opérations arithmétiques

- Opérations de base : `+ - * /`
- Modulo : `%`
- Attention à la division entière :
  - 1/2 vaut 0
  - 1/2.0 vaut 0.5

Réels particuliers :

- Double.NaN
- Double.POSITIVE_INFINITY

Opérations arithmétiques

- Opérations de base : `+ - * /`
- Modulo : `%`
- Attention à la division entière :
  - 1/2 vaut 0
  - 1/2.0 vaut 0.5

Réels particuliers :

- Double.NaN
- Double.POSITIVE_INFINITY
- Comportement défini par IEEE 754

Opérations arithmétiques

X	Y	X/Y	X%Y
valeur finie	0	$+\infty$	NaN
valeur finie	$\pm\infty$	0	x
0	0	NaN	NaN
$\pm\infty$	valeur finie	$\pm\infty$	NaN
$\pm\infty$	$\pm\infty$	NaN	NaN

Opérateurs sur les bits

Opérateur	Exemple	Signification
`<<`	`a << n`	décalage à gauche de *n* bits
`>>`	`a >> n`	décalage à droite de *n* bits
`>>>`	`a >>> n`	décalage à droite non signé de *n* bits
`~`	`~a`	NON binaire
`&`	`a & b`	ET binaire
`^`	`a ^ b`	OU exclusif binaire
`|`	`a | b`	OU binaire

Priorité des opérateurs

les parenthèses	( )
les opérateurs d'incrémentation	++, - -
les opérateurs de multiplication, division, et modulo	*, /,%
les opérateurs d'addition et soustraction	+,-
les opérateurs de décalage	<<,>>
les opérateurs de comparaison	<,>,<=,>=
les opérateurs d'égalité	==, !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	|
l'opérateur ET logique	&&
l'opérateur OU logique	||
les opérateurs d'affectation	=,+=,-=

Sommaire

1 Notion de variable
Déclaration
Types
Portée
Opérations

2 **Instructions du langage Java**
Commentaires
Littéraux (constantes)
Mots clés
Branchements conditionnels
Boucles

3 Premiers programmes

Règles de base

- Java est sensible à la casse

Règles de base

- Java est sensible à la casse
- les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ' ;'

Règles de base

- Java est sensible à la casse
- les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ' ;'
- une instruction peut tenir sur plusieurs lignes

Règles de base

- Java est sensible à la casse
- les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ' ;'
- une instruction peut tenir sur plusieurs lignes
- l'indentation est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur

Sommaire

1 Notion de variable
Déclaration
Types
Portée
Opérations

2 **Instructions du langage Java**
Commentaires
Littéraux (constantes)
Mots clés
Branchements conditionnels
Boucles

3 Premiers programmes

Commentaires

- Commentaires : remarques en langage naturel

Commentaires

- Commentaires : remarques en langage naturel
- Ignorés par le compilateur

Commentaires

- Commentaires : remarques en langage naturel
- Ignorés par le compilateur
- Trois types de commentaires :

Commentaires

- Commentaires : remarques en langage naturel
- Ignorés par le compilateur
- Trois types de commentaires :

Exemple (Commentaire sur une ligne)

```
// zone de declaration des variables
double x; // abscisse
```

Commentaires

- Commentaires : remarques en langage naturel
- Ignorés par le compilateur
- Trois types de commentaires :

Exemple (Commentaire sur plusieurs ligne)

```
/* debut de commentaire
 * suite du commentaire
 * et fin du commentaire */
```

Commentaires

- Commentaires : remarques en langage naturel
- Ignorés par le compilateur
- Trois types de commentaires :

Exemple (Documentation automatique)

```
/**
 * commentaire de la methode
 * @param val la valeur a traiter
 * @since 1.0
 * @return Rien
 * @deprecated Utiliser la nouvelle methode XXX
 */
```

Sommaire

1 Notion de variable
Déclaration
Types
Portée
Opérations

2 **Instructions du langage Java**
Commentaires
Littéraux (constantes)
Mots clés
Branchements conditionnels
Boucles

3 Premiers programmes

Types de littéraux

- Entiers : base 10

Exemple (Littéral)

42, -7

Types de littéraux

- Entiers : base 10 ,base 16

Exemple (Littéral)

0x9AE3

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8

Exemple (Littéral)

020 (= 16)

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8
- Réels : double précision

Exemple (Littéral)

3.14, 6.02E23 ($= 6.02.10^{23}$)

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8
- Réels : double précision simple précision

Exemple (Littéral)

1.5f

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8
- Réels : double précision simple précision
- Booléens

Exemple (Littéral)

true, false

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8
- Réels : double précision simple précision
- Booléens
- Caractères

Exemple (Littéral)

'a', ' ;'

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8
- Réels : double précision simple précision
- Booléens
- Caractères unicode

Exemple (Littéral)

'\u0065' (='e')

Types de littéraux

- Entiers : base 10 ,base 16 ,base 8
- Réels : double précision simple précision
- Booléens
- Caractères unicode
- Chaînes de caractères

Exemple (Littéral)

"Une chaîne de caractères"

Sommaire

1 Notion de variable
Déclaration
Types
Portée
Opérations

2 **Instructions du langage Java**
Commentaires
Littéraux (constantes)
Mots clés
Branchements conditionnels
Boucles

3 Premiers programmes

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	false	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	null	package	private	protected
public	return	short	static	super
switch	synchronized	this	throw	throws
transient	true	try	void	volatile
while				

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	false	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	null	package	private	protected
public	return	short	static	super
switch	synchronized	this	throw	throws
transient	true	try	void	volatile
while				

Sommaire

1 Notion de variable
Déclaration
Types
Portée
Opérations

2 **Instructions du langage Java**
Commentaires
Littéraux (constantes)
Mots clés
Branchements conditionnels
Boucles

3 Premiers programmes

if

- Teste une condition booléenne
- Si **vrai**, exécute une partie du code
- Sinon, exécute autre partie

if

- Teste une condition booléenne
- Si **vrai**, exécute une partie du code
- Sinon, exécute autre partie

Exemple (if)

```
if (expressionBooleenne1) {
     action1
} else if (expressionBooleenne2) {
     action2
} else {
     action3
}
```

Exemple de test

Exemple

```
boolean b = fonctionTest();
int i = calculComplique();
if ( (i>42) && b) {
     appelFonction();
}
```

Remarque

- **if** peut s'écrire sans **else**

Remarque

- **if** peut s'écrire sans **else**

Correct mais à proscrire

```
if (condition) {
} else {
     action;
}
```

Remarque

- **if** peut s'écrire sans **else**

Correct mais à proscrire

```
if (condition) {
} else {
     action;
}
```

Forme préférée

```
if (! condition) {
     action;
}
```

switch/case

Exemple (switch)

```
int i = genererUnEntier();
switch(i) {
case 0:
     liste d'instructions...
     break;
case 1:
     autres instructions...
     break;
default:
     instructions differentes...
}
```

Remarque

- Variable du **switch** de type **int** ou convertie implicitement en **int** : **byte, short, char**
- Instruction **break** : ne pas traiter les autres cas
- Possibilité de regrouper les instructions pour plusieurs cas

Remarque

- Variable du **switch** de type **int** ou convertie implicitement en **int** : **byte, short, char**
- Instruction **break** : ne pas traiter les autres cas
- Possibilité de regrouper les instructions pour plusieurs cas

Exemple

```
switch(i) {
case 0: case 2: case 4:
     ... break;
case 1: case 3:
     ... break;
}
```

Opérateur ?

Opérateur ternaire :

```
( condition )? valeur-vrai : valeur-faux
```

Opérateur ?

Opérateur ternaire :

```
( condition )? valeur-vrai : valeur-faux
```

Exemple

```
// i et j sont des entiers
int max;
max = (i>j)? i : j;
```

Sommaire

1 Notion de variable
Déclaration
Types
Portée
Opérations

2 **Instructions du langage Java**
Commentaires
Littéraux (constantes)
Mots clés
Branchements conditionnels
Boucles

3 Premiers programmes

Principe

- Boucle = structure de contrôle
- But : exécuter certaines opérations plusieurs fois
- Il existe plusieurs types de boucles en Java
- Toutes les boucles sont équivalentes

while

```
while(condition) {
     actions
}
```

while

```
while(condition) {
     actions
}
```

Exemple (boucle while)

```
int i=0;
while (i<10) // pas de ";"
{
     System.out.println(i);
     i++; // i=i+1
}
```

do-while

Exemple

```
char c;
do {
     c = lireUnCaractereAuClavier();
}  while (c != ' '); // ici ";"
```

Différence avec le **while** : le bloc est exécuté au moins une fois.

for

Utilisation : le nombre d'itérations est connu. Syntaxe :

```
for (initialisation ; condition ; modification)
```

for

Utilisation : le nombre d'itérations est connu. Syntaxe :

```
for (initialisation ; condition ; modification)
```

- **initialisation** exécuté avant de commencer la boucle ;

for

Utilisation : le nombre d'itérations est connu. Syntaxe :

```
for (initialisation ; condition ; modification)
```

- **initialisation** exécuté avant de commencer la boucle ;
- **condition** détermine s'il est possible de continuer la boucle ;

for

Utilisation : le nombre d'itérations est connu. Syntaxe :

```
for (initialisation ; condition ; modification)
```

- **initialisation** exécuté avant de commencer la boucle ;
- **condition** détermine s'il est possible de continuer la boucle ;
- **modification** modifie les données pour l'itération suivante.

for

Exemple (Équivalent du while)

```
int i;
for (i=0 ; i<10 ; i++) {
     System.out.println(i);
}
```

for

Exemple (Équivalent du while)

```
for (int i=0 ; i<10 ; i++) {
     System.out.println(i);
}
```

Remarque

Possibilité de déclarer l'indice de boucle lors de l'initialisation
$\Rightarrow$ variable locale à la boucle

Sortie de boucle

Possibilité de modifier l'exécution d'une boucle avec **break** et **continue**. Mais ne pas en abuser.

Exemple

```
while( condition1 ) {
     if (condition2)
          break;
     ...
}
```

Sortie de boucle

Possibilité de modifier l'exécution d'une boucle avec **break** et **continue**. Mais ne pas en abuser.

Exemple

```
while (condition1 && !condition2) {
      ...
}
```

Sommaire

1 Notion de variable
- Déclaration
- Types
- Portée
- Opérations

2 Instructions du langage Java
- Commentaires
- Littéraux (constantes)
- Mots clés
- Branchements conditionnels
- Boucles

3 Premiers programmes

Hello World

```
/**
 * Premier programme Java : hello world
 */
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello");
  }
}
```

Remarques

- Nom du fichier et nom du programme doivent coïncider
- Ex : le fichier doit s'appeler **Hello.java**
- Compilation : **javac Hello.java**
- Exécution : **java Hello**

Passage de paramètres au main

- Passage de paramètres par la variable **args**
- Ex : **java Hello 1234 xyz**
  - args[0] $\leftarrow$ 1234
  - args[1] $\leftarrow$ xyz