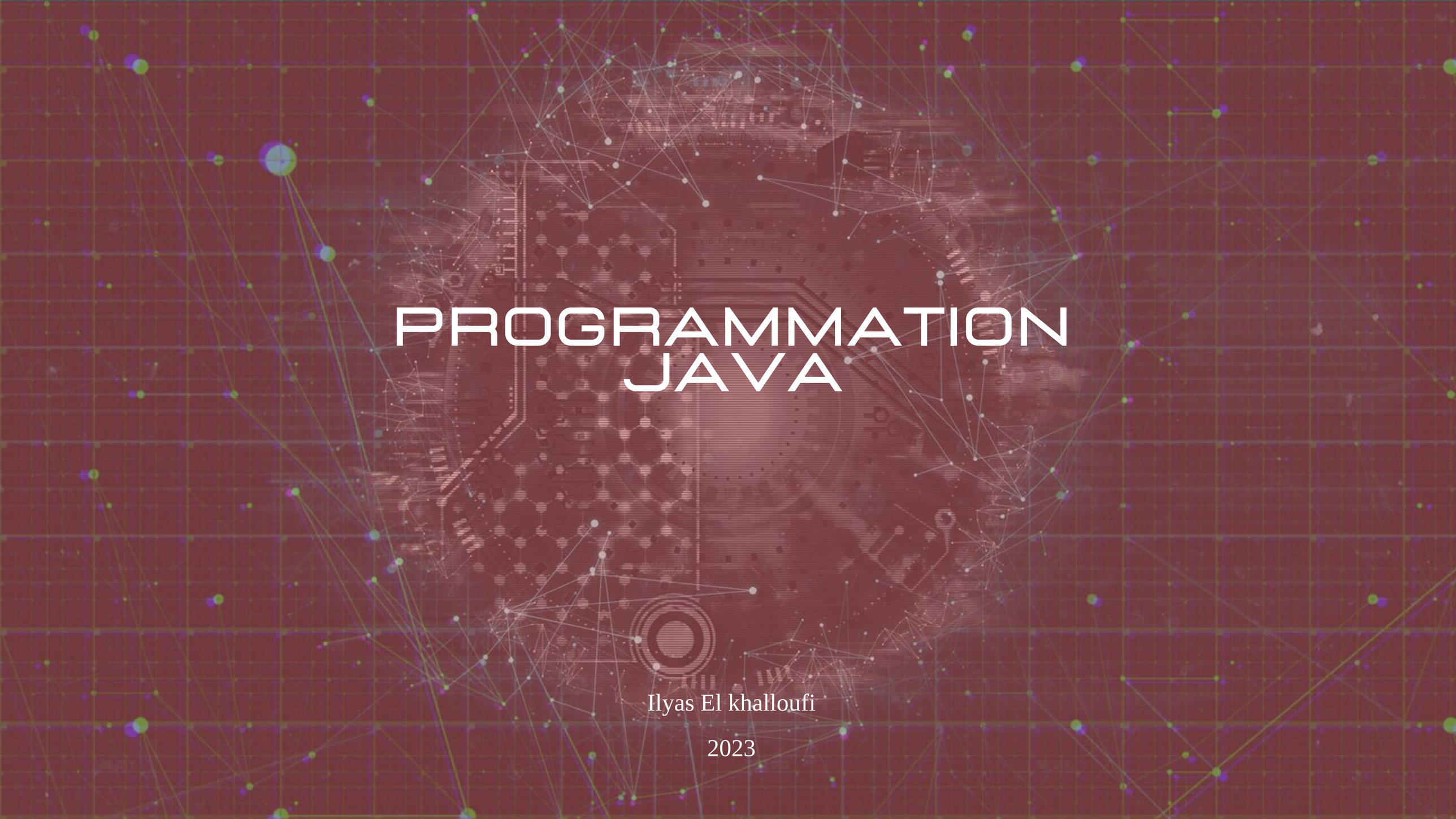


PROGRAMMATION JAVA

Ilyas El khalloufi

2023

PROGRAMME DE LA FORMATION

Module 1 : Introduction à JAVA

- Présentation de JAVA
- Premier programme
- Variables et types de données

Module 2 : Programmation orientée objet

- Définitions
- Classes/Méthodes/Attributs
- Principes d’encapsulation

Module 3 : Structures de contrôles

- Conditions
- Boucles
- Exercices pratiques

Module 4 : Fonctions et méthodes

- Définitions
- Passage de paramètres
- Cas d’application

Module 5 : Cas pratiques

MODULE 1
INTRODUCTION

INTRODUCTION

PRESENTATION DU LANGUAGE

- Installation des outils : Eclipse org

- JAVA est un langage orienté objet développé initialement par la société Sun puis par Oracle.

- Les avantages de JAVA sont nombreux :
  - Gratuité
  - Communauté
  - Portabilité
  - Pérennité

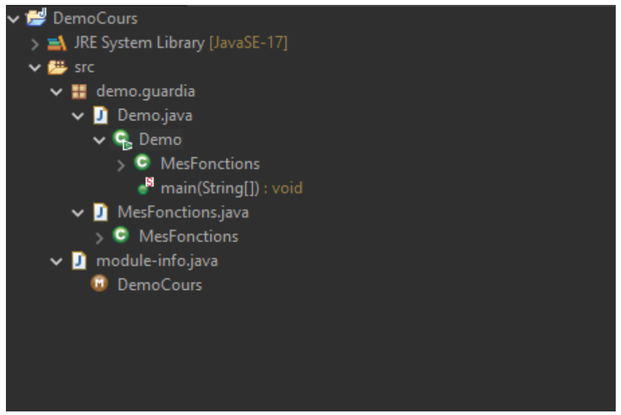
- JAVA est utilisé dans de nombreux domaines (du WEB à l'embarqué) et dispose de plusieurs distributions (E, ME, EE, Android).

- Le programme s'exécute dans une machine virtuelle (JVM – Java Virtual Machine)

INTRODUCTION

PRESENTATION DU LANGUAGE

- JAVA produit du code compilé
- Pour créer un programme, on fonctionne en trois étapes : Ecriture, Compilation et Exécution.
- Structure de base d’un programme :




```java
package demo.guardia;


public class Demo {

    public class MesFonctions {
        public int somme(int a, int b) {
            return a+b;
        }
    }

    public static void main(String[] args) {

        Demo obj = new Demo();
        Demo.MesFonctions objMesFonctions = obj.new MesFonctions();

        System.out.println("Hello world!");

        for(int i=0; i<10; ++i) {
            System.out.printf("[%d]\n", i);
        }

        int a = 1;
        int b = 1;
        int s = objMesFonctions.somme(a, b);
        System.out.printf("%d + %d = %d", a,b,s);
    }
}
```

INTRODUCTION

PRESENTATION DU LANGUAGE : TYPES

Entiers signés (+/-) et non signés (+)

- short
  - >= 2 octets
  - Taille minimale de 16 bits garantie par le standard
  - Signé
  - Valeur entre -32 768 et 32 767 garanties

- signed short
  - Equivalent à short

- unsigned short
  - Equivalent à short
  - Non signé
  - Valeur entre 0 et 65 535 garanties

Entiers signés (+/-) et non signés (+)

- int
  - >= 2 octets
  - Taille minimale de 16 bits garantie par le standard
  - Signé
  - Valeur entre -32 768 et 32 767 garanties

- signed int
  - Equivalent à int

- unsigned int
  - Equivalent à int
  - Non signé
  - Valeur entre 0 et 65 535

Entiers signés (+/-) et non signés (+)

- long
  - >= 4 octets
  - Taille minimale de 32 bits garantie par le standard
  - Signé
  - Valeur entre -2 147 483 648 et 2 147 483 647 garanties

- signed long
  - Equivalent à long

- unsigned long
  - Equivalent à long
  - Non signé
  - Valeur entre 0 et 4 294 967 295

Entiers signés (+/-) et non signés (+)

long long
>= 8 octets
Taille minimale de 64 bits garantie par le standard
Signé
Valeur entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807 garanties

signed long long
Equivalent à long long

unsigned long long
Equivalent à long long
Non signé
Valeur entre 0 et 18 446 744 073 709 551 615

INTRODUCTION

PRESENTATION DU LANGUAGE : TYPES

Virgules flottantes

- float
  - >= 4 octets
  - Taille minimale de 32 bits garantie par le standard
  - Valeur entre $-3.4*10^{-38}$ à $3.4*10^{38}$ garantie avec une précision 7 chiffres significatifs
  - Souvent au format IEEE-754 si supporté

- double
  - >= 8 octets
  - Taille minimale de 64 bits garantie par le standard
  - Valeur entre $-1.7*10^{-308}$ à $1.7*10^{308}$ garantie avec une précision 7 chiffres significatifs
  - Souvent au format IEEE-754 si supporté

- long double
  - Taille et précision non garantie par le standard
    - » Au moins la même taille et précision que double
  - Souvent identique a double ou format 128 bits

Booléen

bool

1 octets
Taille fixe de 8 bits garantie par le standard
true défini par le standard à != 0
false défini par le standard à == 0

char

>= 1 octet
Taille minimale de 8 bits garantie par le standard
Signé ou non signé (Représentation dépendante du compilateur)
Uniquement pour les caractères

INTRODUCTION

PRESENTATION DU LANGUAGE : MOTS CLES

abstract		class		false		instanceof		private		switch		void
assert	(SE 1.4)	continue		final		int		protected		synchronized		volatile
boolean		default		finally		interface		public		this		while
break		do		float		long		return		throw		
byte		double		for		native		short		throws		
case		else		if		new		static		transient		
catch		enum	(SE 5.0)	implements		null		strictfp	(SE 1.2)	try		
char		extends		import		package		super		true		

exports	(SE 9.0)	open	(SE 9.0)	provides	(SE 9.0)	sealed	(SE 17.0)	uses	(SE 9.0)	yield	(SE 14.0)
module	(SE 9.0)	opens	(SE 9.0)	record	(SE 16.0)	to	(SE 9.0)	var	(SE 10.0)		
non-sealed	(SE 17.0)	permits	(SE 17.0)	requires	(SE 9.0)	transitive	(SE 9.0)	with	(SE 9.0)		

MODULE 2

PROGRAMMATION ORIENTEE OBJET

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : OBJETS

- Les objets sont des instanciations de classes, de structures ou d'unions

```
[class/struct/union] S
{
};

// objet est une instance de S
S objet;
```

- Un champ est appelé attribut de classe ou attribut de structure

```
[class/struct/union] S
{
    int attribut_1;
};
```

- Un attribut peut être statique ou membre

```
[class/struct/union] S
{
    // Attribut membre
    int attribut_1;

    // Attribut statique ( Déclaration )
    static int attribut_2;
};

// Définition obligatoire
int S::attribut_2;
```

- Un attribut membre doit être accédé avec un objet

```
[class/struct/union] S
{
    // Attribut membre
    int attribut_1;
};

S objet;

// Lecture de l'attribut membre de objet
int valeur = objet.attribut_1;
```

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : ATTRIBUTS

- Un champ peut être accessible ou non

```
[class/struct/union] S
{
public: // Accessible à partir d'ici
    int attribut_1; // Modifiable

private: // Non accessible à partir d'ici
    int attribut_3;

protected: // Non accessible à partir d'ici ( Sauf enfants directs )
    int attribut_4;
};
```

- Un champ peut être modifiable ou non

```
[class/struct/union] S
{
public: // Accessible à partir d'ici
    int attribut_1; // Modifiable
    const int attribut_2; // Non modifiable

private: // Non accessible à partir d'ici
    int attribut_3;

protected: // Non accessible à partir d'ici ( Sauf enfants direct )
    int attribut_4;
};
```

- Une fonction est appelée fonction de classe ou fonction de structure

```
[class/struct/union] S
{
    int fonction();
};
```

- Une fonction peut être statique ou membre

```
[class/struct/union] S
{
    // Fonction membre
    int fonction_membre();

    // Fonction statique
    static int fonction_statique();
};
```

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : METHODES

- Une fonction membre doit être accédée avec un objet
- Une fonction membre à accès à tous les attributs membres de sa classe

```
[class/struct/union] S
{
    // Fonction membre
    int fonction_membre();
};

S objet;

// Appel de la fonction membre de objet
int valeur = objet.fonction_membre();
```

- Une fonction statique n'a pas besoin d'objet pour être appelée
- Une fonction statique n'a accès qu'aux membres statiques de sa classe

```
[class/struct/union] S
{
    // Fonction statique
    static int fonction_statique();
};

// Appel de la fonction statique de la classe S
int valeur = S::fonction_statique();
```

- Une fonction membre peut être accessible publiquement ou non

```
[class/struct/union] S
{

public: // Accessible en dessous

    int fonction();

private: // Non accessible

    int fonction_2();

protected: // Non accessible ( Sauf enfants directs )

    int fonction_3();
};
```

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : CONSTRUCTEUR

- Une fonction membre spéciale
  - Appelée 1 fois à la création d'un objet de la classe
  - Les règles de visibilité s'appliquent aux constructeurs

```
[class/struct/union] S
{

[public/private/protected] :

    // Construit un objet S et initialise ses attributs
    S() {
        attribut_1 = 1;
    }

    int attribut_1;
};
```

- Un constructeur '*par défaut*' est généré automatiquement par le compilateur si aucun autre constructeur n'est défini

- Un constructeur est dit '*par défaut*' si il n'a aucun paramètre

```
[class/struct/union] S
{

[public/private/protected] :

    // Constructeur par défaut
    S() {
        attribut_1 = 1;
    }

    int attribut_1;
};
```

- Un constructeur est dit '*paramétrique*' si il a des paramètres

```
[class/struct/union] S
{

[public/private/protected] :

    // Constructeur paramétrique
    S(int valeur) {
        attribut_1 = valeur;
    }

    int attribut_1;
};
```

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : LE DESCTRUCTEUR

- Une fonction membre spéciale
  - Appelée 1 fois à la destruction d'un objet de la classe
  - Les règles de visibilité s'appliquent aux destructeurs mais 99% du temps il doit être public

```
[class/struct/union] S
{
[public/private/protected] :

    // Détruit un objet S et détruit ses attributs
    ~S() {
    }
};
```

- Un destructeur public est généré par défaut

```
[class/struct/union] S
{
    // Constructeur public généré par défaut
};
```

- Un destructeur détruit les attributs membres dans l'ordre inverse de leurs déclarations

```
[class/struct/union] S
{
    // Détruit après attribut_2
    int attribut_1;

    // Détruit en premier
    int attribut_2;
};
```

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : VISIBILITE

- public : Accessible à l'extérieur de la class ou struct

```
[class/struct] Vehicule {
    public: // Accessible dans main et Voiture
    Marque marque;
};

[class/struct] Voiture : Vehicule{
    Marque const& get_marque() const{ return marque; }
};

int main(){
    Voiture voiture = Voiture{};
    Marque& marque = voiture.marque; // Ok
    marque = voiture.get_marque(); // Ok
}
```

- protected : Accessible uniquement par les class ou struct enfants

```
[class/struct] Vehicule {
    protected: // Accessible dans Voiture
    Marque marque;
};

[class/struct] Voiture : Vehicule{
    Marque const& get_marque() const{ return marque; }
};

int main(){
    Voiture voiture = Voiture{};
    Marque const& marque = voiture.marque; // Nok
    Marque const& marque = voiture.get_marque(); // Ok
}
```

PROGRAMMATION ORIENTEE OBJET

CONCEPTS : VISIBILITE

- private : Accessible uniquement par la class ou la struct

```
[class/struct] Vehicule {
    private: // Accessible uniquement dans Vehicule
    Marque marque;
};

[class/struct] Voiture : Vehicule{
    Marque const& get_marque() const{ return marque; } // Nok
};

int main(){
    Voiture voiture = Voiture{};
    Marque const& marque = voiture.marque; // Nok
    Marque const& marque = voiture.get_marque(); // Nok
}
```

MODULE 3

STRUCTURES DE CONTROLES

PROGRAMME DE LA FORMATION

Module 1 : Introduction à JAVA

- Présentation de JAVA
- Premier programme
- Variables et types de données

Module 2 : Programmation orientée objet

- Définitions
- Classes/Méthodes/Attributs
- Principes d’encapsulation

Module 3 : Structures de contrôles

- Conditions
- Boucles
- Exercices pratiques

Module 4 : Fonctions et méthodes

- Définitions
- Passage de paramètres
- Cas d’application

Module 5 : Cas pratiques

STRUCTURES DE CONTROLES

IF_ELIF_ELSE

- Permet de réaliser des actions conditionnelles :

```
if(s == 2) {
    System.out.printf("s vaut 2");
}
else if (s ==3) {
    System.out.printf("s vaut 3");
}
else {
    System.out.printf("s est different de 2 et de 3");
}
```

Opérateur de comparaisons

- < strictement inférieur
- > strictement supérieur
- <= inférieur ou égal
- >= supérieur ou égal
- == égal
- != différent
- <> différent, on utilisera de préférence !=
- X is Y : X et Y représentent le même objet.
- X is not Y : X et Y ne représentent pas le même objet

STRUCTURES DE CONTROLES

SWITCH_CASE

- Permet de réaliser des actions conditionnelles :

```
1 public class Demo {
2
3     public static void main( String [] args ) {
4
5         double value = Math.random() * 10;
6         switch ( value ) {
7             case 1.0:
8                 System.out.println( "Ne peut pas compiler" );
9                 break;
10            default:
11                System.out.println( "Autre valeur" );
12        }
13
14    }
15
16 }
```

- Chaque case est associé à une valeur à tester et il se termine par une instruction break. Cette dernière instruction est très importante dans notre cas : elle permet que l'exécution ne se poursuive pas au case suivant.

STRUCTURES DE CONTROLES

FOR

- Permet de réaliser des actions itératives :

```java
public class Demo {

    public static void main( String [] args ) {

        for ( int counter = 1; counter <= 10; counter++ )
            System.out.println( counter );

        System.out.println( "Bye bye" );
    }

}
```

STRUCTURES DE CONTROLES

WHILE

- Permet de réaliser des actions itératives sous condition :

```java
public class Demo {

    public static void main( String [] args ) {

        // La première boucle est réalisée via l'instruction for

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

        // La seconde boucle est réalisée via l'instruction while

        int i=0;
        while( i<10 ) {
            System.out.print( i + " " );
            i++;
        }
        System.out.println();

    }

}
```

STRUCTURES DE CONTROLES

BREAK/CONTINUE

- Break permet d'interrompre une boucle :

```java
public class Demo {

    public static void main( String [] args ) {

        for( int i=0; i<10; i++ ) {
            // On teste si on doit quitter la boucle
            if ( i == 5 ) break;

            // On affiche la valeur de la variable i
            System.out.println( "i == " + i );
        }

    }
```

- Continue permet d'interrompre une itération d'une boucle :

```java
public class Demo {

    public static void main(String args[]){
        int sum = 0;

        for( int i=1; i<=10; i++ ) {
            if ( i == 5 ) continue;
            sum += i;
        }

        String message = "La somme des dix premiers entiers positifs (privé de 5) est de : ";
        System.out.println( message + sum );
    }

}
```

MODULE 4

FONCTIONS ET METHODES

FONCTIONS ET METHODES

- Les fonctions en Java sont appelées "méthodes". Une méthode est un bloc de code réutilisable qui effectue une tâche spécifique et peut prendre des paramètres en entrée, effectuer des calculs, et renvoyer un résultat. Les méthodes sont définies à l'intérieur des classes et peuvent être invoquées (appelées) depuis d'autres parties du code.

```
<type_de_retour> nom_de_la_methode(<parametres>) {
    // Corps de la méthode
    // Code pour effectuer une tâche
    // return <valeur>; (si la méthode renvoie une valeur)
}
```

- Les fonctions permettent de rendre le code plus modulaire.
- Une fonction qui ne retourne pas de valeur est une **procédure**.
- Une fonction dans une classe est une **méthode**.

MODULE 5
CAS PRATIQUES

EXERCICES D'APPLICATIONS

CAS PRATIQUES NIVEAU 1

Exercice 1 – entrée/sortie dans un programme

Ecrire un algorithme qui affiche la somme de deux entiers positifs saisis au clavier.

Exercice 2 – Détection d'une valeur dans un tableau

Soit un tableau de 10 entiers T = {5, 5, 7, -1, 0, 0 ,0 ,0 ,0, 0}. La valeur -1 termine une série d'éléments non nuls.

Proposer deux algorithmes qui calculent la position (l'index) de l'élément -1 dans ce tableau. Le premier s'appuiera sur une boucle TANT QUE…FAIRE…, le second utilisera une boucle POUR … ALLANT DE … A … FAIRE…

Exercice 3 – Instruction CASOU

Ecrire un algorithme qui affiche le résultat de calcul (+, -, ×, /) de deux entiers positifs saisis au clavier. Dans cet exercice, on utilisera une structure CASOU pour choisir l'opérateur à l'aide d'un affichage à l'écran et d'une saisie au clavier.

Exercice 4 – Calcul statistique

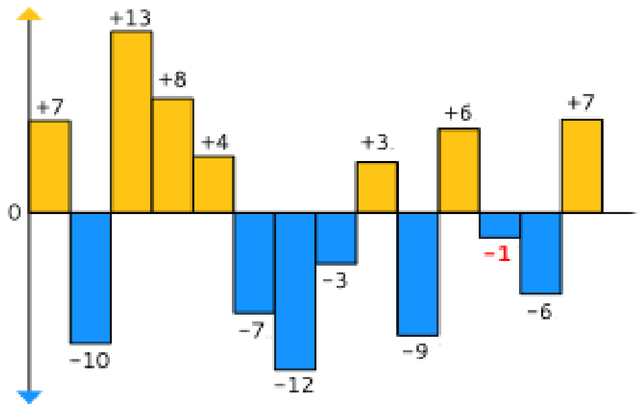
Soit le tableau de réels T = {5.0, 5.5, 3.0, 7.0, 8.0, 4.0, 6.0, 9.0, 3.3, 5.0}.

Proposer un algorithme qui affiche la somme, la moyenne, la plus petite valeur et la plus grande valeur de la série.

EXERCICES D'APPLICATIONS

CAS PRATIQUES NIVEAU 2

Cas 1 [FACILE] – Dans cet exercice, vous devez analyser un relevé de température pour trouver quelle température se rapproche le plus de zéro.



Ecrivez un algorithme qui affiche la température la plus proche de 0 parmi les données d'entrée. Si deux nombres sont aussi proches de zéro, alors l'entier positif sera considéré comme étant le plus proche de zéro (par exemple, si les températures sont -5 et 5, alors afficher 5).

Entrées : Votre algorithme doit lire les données depuis le clavier et afficher le résultat à l'écran.

Sortie : Affichez 0 (zéro) si aucune température n'est fournie. Sinon affichez la température la plus proche de 0.

Exemple : Entrées : 5, 1, -2, -8, 4, 5 → Sortie : 1

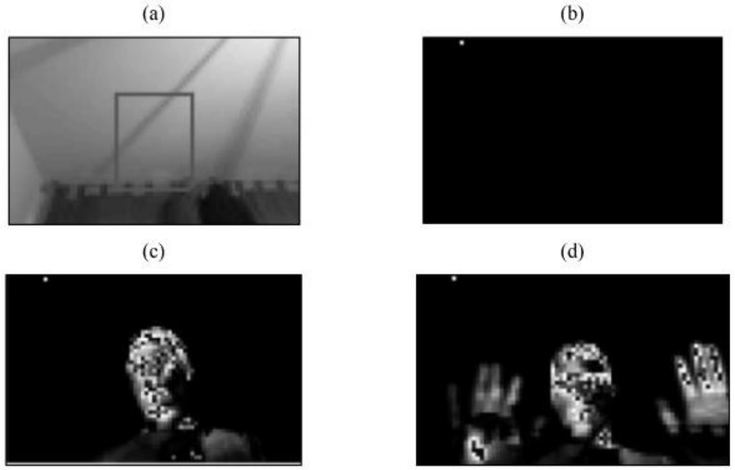
EXERCICES D'APPLICATIONS

CAS PRATIQUES NIVEAU 2

Cas 2 [MOYEN] - Dans cet exercice, vous devez créer un système de détection de mouvement à partir d'une caméra statique.

En général, il est courant de rencontrer des systèmes de détections de mouvements par télémétrie laser. Ces systèmes envoient un faisceau laser dont le temps de vol permet de convertir la position d'une cible en distance. Dès lors que ce faisceau est coupé, par une personne par exemple, le temps de vol est beaucoup moins important étant donné que la réflexion du faisceau se fait sur la personne et non plus sur la cible. On peut dès lors identifier le passage d'une personne.

Une autre approche consiste à positionner une caméra qui va réaliser un différentiel d'images entre un instant t et l'instant d'après $t + 1$ seconde par exemple. S'il n'y a aucun mouvement tous les pixels de l'image vont s'annuler. S'il y a du mouvement, les pixels ne s'annuleront pas et on aura la capacité de détecter le mouvement.



Entrées : Votre algorithme lit les deux images successives dans une base de données.

Sortie : L'algorithme affiche à l'écran le premier couple de coordonnées (x, y) où le mouvement a été détecté.

EXERCICES D'APPLICATIONS

CAS PRATIQUE NIVEAU 3

Cas 3 [DIFFICILE] - Dans cet exercice, vous devez créer un système de détection de mouvement à partir d'une caméra statique.

En repartant du système du cas 2, proposer un algorithme qui permet de suivre le mouvement de la cible sur une séquence de 10 images.

Entrées : Votre algorithme lit les 10 images successives dans une base de données.

Sortie : L'algorithme affiche à l'écran une suite de coordonnées (x, y) correspondant à la position de la cible dans chaque image.

EXERCICES D'APPLICATIONS

MINI PROJET 1

Les robots autonomes sont monnaie courante de nos jours. Certains arrivent même à se déplacer sans aucune connaissance de l'environnement qui l'entoure. L'objectif de notre projet consiste à développer un algorithme de planification permettant à un robot autonome de se déplacer d'un point A vers un point B en évitant tous les obstacles.

Entrée : Une matrice d'environnement contenant les potentiels (les coefficients) de la carte de déplacement. Une position de départ en rouge et une position d'arrivée en vert clair.

136	136	500	136	136	136	136	136	136	136	136	136	136	136	136	136	136	136	136	1
136	120	120	120	120	500	120	120	120	120	120	120	120	120	120	120	120	120	120	2
136	120	105	105	105	105	105	500	105	105	105	105	105	105	105	105	500	105	105	3
136	120	105	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	4
136	120	105	500	78	78	78	78	78	78	78	78	500	78	500	78	78	78	78	5
500	120	105	91	78	66	66	66	66	500	66	66	66	66	66	66	66	66	66	6
136	120	105	91	78	66	55	55	55	55	55	55	55	55	55	55	55	55	500	7
136	500	500	91	500	66	55	45	45	45	45	45	45	45	45	45	45	45	45	8
136	120	105	91	78	66	55	500	36	36	36	36	36	36	36	500	36	36	36	9
136	120	105	91	78	66	55	45	36	28	28	28	28	28	28	28	28	28	28	10
136	120	105	91	500	66	55	500	36	28	500	21	21	21	21	21	21	21	500	11
136	120	500	91	78	66	55	45	36	28	21	15	15	15	15	15	15	15	15	12
136	120	105	91	78	66	55	45	36	28	21	15	10	10	10	10	10	10	10	13
136	120	105	500	78	66	500	45	36	28	21	15	10	500	6	6	6	6	6	14
136	120	105	91	78	66	55	45	36	28	21	15	10	6	3	3	3	3	500	15
136	500	105	91	78	66	55	500	36	500	21	15	10	6	3	1	1	1	3	16
136	120	105	91	78	66	55	45	36	28	21	15	10	6	3	1	0	1	3	17
136	120	105	91	78	66	55	45	36	28	21	15	500	6	3	500	1	1	3	18
136	120	105	91	78	66	55	45	36	28	500	15	10	6	3	3	3	3	500	19
136	120	105	91	78	66	55	45	36	28	21	15	10	6	6	6	6	6	6	20
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Sortie : Une séquence de déplacement (vert foncé) du robot d'un point A vers un point B affichée à l'écran. Les directions sont :

DGS	H	DDS
G		D
DGI	B	DDI

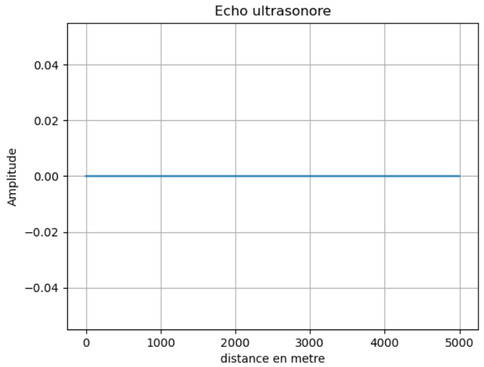
EXERCICES D'APPLICATIONS

MINI PROJET 2

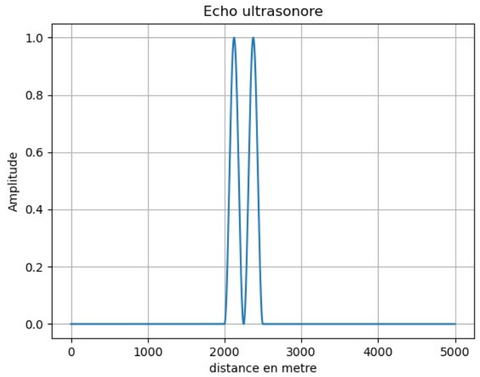
Le temps de calcul est une donnée importante de la majeure partie des systèmes fonctionnant en temps réel. Par exemple, les véhicules actuels disposent, pour la plupart, de télémètre permettant d'activer un freinage d'urgence dans le cas où un danger s'approche.

Le principe de ce système consiste à propulser une impulsion laser en direction de l'avant puis à mesurer la distance avec la cible la plus proche en mesurant une augmentation d'énergie.

Si aucun obstacle n'est présent, le signal mesuré est de la forme suivante :



Si un obstacle est présent, le signal contient de l'énergie qui correspond à l'écho qui s'est réfléchit sur la cible :



Entrée : Un fichier contenant les valeurs associées au signal d'entrée.

Sortie : La position de l'obstacle s'il existe, sinon un message « pas d'obstacle » affiché à l'écran.

EXERCICES D'APPLICATIONS

MINI PROJET 3

Je souhaite réaliser un outil en ligne de commande permettant de monitorer l'état des fichiers sur un disque dur et de les manipuler.

L'outil met à disposition les fonctions suivantes :

- Scan <nom du dossier/disque> : réalise le scan complet du disque, affiche le nombre de fichiers et le pourcentage de consommation mémoire.
- Copy <fichier A> <destination> : Copie le fichier A dans le répertoire de destination.
- Show <fichier.txt> : Affiche le contenu du fichier sur la console.
- Delete <fichier B> : Supprime le fichier B.
- Help : Affiche les fonctions (ainsi que leur paramètres) dans la console.
- Stat <nom du dossier/disque>: Afficher le nom, le chemin et la taille du plus gros fichier présent dans l'arborescence pointée.