Java : Aide mémoire

Téléchargement et ressources

http://openjdk.java.net/ (**JDK**= Java Developpement Kit à décompresser dans c:\Program Files)
https://netbeans.apache.org/ (EDI = Environnement de développement intégré à installer)
https://www.w3schools.com/java/default.asp et https://www.jmdoudoux.fr/java/dej/index.htm
Sous Windows ajoutez le **path** (vers dossier bin du JDK) et **JAVA_HOME** (vers JDK)

Compiler et exécuter

```
javac Essai.java (Compile le programme java)
java Essai (Exécute un programme)
Ctrl C (Interrompt un programme en mode console)
java -version (donne la version du JDK utilisé)
```

Un exemple exécutable

```
package essai;
public class essai {
	public static void main(String[] args) { // main() est static
		/* Commentaire multi-ligne */
		final int NUM_VERSION = 17; // constante
		String message = "Bonjour, java numéro "+ NUM_VERSION +" ?";
		// Commentaire sur une ligne ou fin de ligne
		System.out.println(message);
}	}
```

Types de variable

int	float	char	boolean	String
byte	short	long	double	[ ] (voir tableau)

Tableau **[ ]** (type_du_contenu[] nom_du_tableau; exemple : **int[] mesNombres;**)

Conversion de type : variable = (type de destination) autreVariable ; **int i = (int) reel;**

Opérateur d'affectation **=** (affecte le membre de droite au membre de gauche)

Opérateurs arithmétiques

+	**-**	*****	**/**	**%** *(modulo)*	
++	**--**	**+=**	**-=**	***=**	**/=**

Concaténation de chaînes + **+=** ajoute (la chaîne de droite à la chaîne de gauche)

Opérateurs logiques

&& (ET logique) **||** (OU logique) **!** (NON logique)
? (**opérateur ternaire**) `variable = (condition) ? valeur_si_vrai : valeur_si_faux;`

Opérateurs de comparaison

== (égalité)	**!=** (inégalité)	**<** (inférieur)
>	**<=**	**>=**

Opérateurs binaires

! (inversion)	**&** (ET)	**|** (OU)	**^** (XOR)
~ (complément)	**<<** (décalage)	**>>**	**>>>**

Fonctions mathématiques

E	PI	abs()	acos()	asin()	atan()	*atan2()*
cbrt()	cos()	cosh()	exp()	floor()	hypot()	log()
log10()	max()	min()	pow()	random()	round()	sin()
sinh()	sqrt()	tan()	tanh()	toDegrees()	toRadians()	...

Les chaînes de caractères

Une chaîne de caractère (type **String**) est délimité par des guillemets doubles **"**
Un caractère (type **char**) est délimité par guillemets simples : **'**

```
String uneChaine = " Truc;Machin;Bidule " ;// une chaîne est un objet en java
```

Quelques méthodes pour les chaînes

Longueur	i = uneChaine.**lenght()**;
Test si est vide	if (uneChaine.**isEmpty()**) uneChaine = "TMB";
Remplacer	String maChaine = uneChaine.**replace(**"Bidule","Chose"**)**;
Remplacer tout	String maChaine = uneChaine.**replaceAll(**"u","o"**)**;
Convertir en tableau	String[ ] tabChaine= uneChaine.**split(";")**;
Extraire une partie	String maChaine = uneChaine.**substring(debut,fin)**;
Convertir en tableau de char	Char[ ] tabChar = uneChaine.**toCharArray()**;
Mettre en minuscule	String maChaine = uneChaine.**toLowerCase()**;
Mettre en majuscule	String maChaine = uneChaine.**toUpperCase()**;

Séquences d'échappement

\'	\"	\\	\n	\t

Structures de contrôle

boucle tant que … faire

```
while ( condition ) {
        // tant que la condition est remplie, faire
}
```

boucle faire … tant que

```
do {
        // faire, tant que la condition est remplie
} while ( condition )
```

boucle pour (initialisation ; condition ; action) faire

```
int i = 0 ;
for (i=100 ; i > 10 ; i--) {
        // faire, tant que la condition est remplie
}
```

les structures si et selon

```
// if (condition) {…} else {…}

if ( condition_1 ) {
        // faire si condition_1 est vraie
} else if (condition 2) {
        // sinon faire si condition_2 est vraie
} else {
        // sinon faire
}
```

```
switch (expression) {
        case const_1 :
                // faire si expression = const_1
                break; // sinon la suite s'exécute
        case const_2 :
                // faire si expression = const_2
                break;
        default : // si auncune condition remplie
}
```

Sortir d'une boucle	**break ;** // quitte immédiatement une boucle
Passer à la suite	**continue ;** // dans une boucle, passe à l'itération suivante

Organisation en package

Le jdk

Le JDK fournit de nombreuses **classes** regroupées en **packages** (répartis en **modules**)
la classe **math** et intégrée au package **java.lang** du module **java.base**
import permet d'accéder aux classes et méthodes des différents packages
java.lang contient les classes de base du langage Java.
java.lang est systématiquement importé dans les sources.

Quelques packages

java.applet	Classes nécessaires pour créer des applets.
java.awt	Interfaces utilisateur, images et graphiques
java.io	Entrées – sorties système, gestion des flux de données
java.net	Développement d'application utilisant les fonctionnalités réseau
java.rmi	Communication entre objets distants
java.security	Sécurisation des accès et des échanges (signatures et certificats)
java.sql	Utiliser des bases de données relationnelles
java.text	Travailler sur des objets de type texte
Java.util	Ensemble d'outils pour gérer les listes, les collections, les dates ...
javax.crypto	Pour le cryptage des données
javax.swing	Développement d'interfaces graphiques

Les sources du JDK contiennent plus de 70 packages et plus de 15000 classes Java

Des tableaux aux collections

Tableaux

```
// un tableau peut contenir des variables mais aussi des objets...
String[] lesNoms = {"Truc","Bidule","Machin","Chose"};
for (String unNom : lesNoms) System.out.println(unNom);
// équivalent à :
for (int i =0; i< lesNoms.length; i++) {
      System.out.println(lesNoms[i]);    }
```

Les collections

ArrayList	Liste assimilable à un tableau dynamique (interface List)
LinkedList	liste ordonnée, doublement chaînée (début, fin)(interface List)
HashMap	Dictionnaire, ensemble clé-valeur (interface Map)
HashSet	Dictionnaire, ensemble clé-valeur (interface Set)
TreeMap	Dictionnaire organisé en arbre donc trié (interface SortedMap)
TreeSet	Dictionnaire organisé en arbre donc trié (interface SortedSet)

Une liste

```
import java.util.ArrayList; // entre la déclaration du package et de la classe
//[...]
ArrayList<String> lesNoms = new ArrayList<>();
lesNoms.add("Truc"); lesNoms.add("Bidule");lesNoms.add("Machin"); lesNoms.add("La chose");
System.out.println(lesNoms.get(0));      // accéder à un élément
lesNoms.set(3,"Chose");    // modifier un élément
for (String unNom : lesNoms) System.out.println(unNom);
// équivalant à :
lesNoms.forEach((var unNom) -> System.out.println(unNom));
lesNoms.clear();      // vider la liste
```

Un dictionnaire

```
import java.util.HashMap;   // entre la déclaration du package et de la classe
//[...]
HashMap<String, String> lesNoms = new HashMap<>();
lesNoms.put("Zin","Zinia TRUC"); lesNoms.put("Bid","Alien Bidule");
for (String cle : lesNoms.keySet()){
      System.out.print(cle + " "+lesNoms.get(cle)+"\n");
} // voir https://www.jmdoudoux.fr/java/dej/chap-collections.htm
```

Un dictionnaire trié

```
import java.util.TreeMap;   // entre la déclaration du package et de la classe
//[...]
TreeMap<String, String> lesNoms = new TreeMap<>();
lesNoms.put("Zin","Zinia TRUC"); lesNoms.put("Bid","Alien Bidule");
lesNoms.remove("Bid");
lesNoms.put("Amon","Dada AmontchSky");
for (String cle : lesNoms.keySet())       {
      System.out.print(cle + " "+lesNoms.get(cle)+"\n");
}
```

Utiliser une base de donnée

Le driver

Doit être adapté au SGBD et ajouté aux bibliothèques (libraries) du projet
MySQL: mysql-connector-java.jar https://dev.mysql.com/downloads/
PostgreSQL: fourni dans le JDK (PostgreSQL JDBC Driver)

Se connecter

```
import java.sql.Connection;  // entre la déclaration du package et de la classe
import java.sql.DriverManager;
//[...]
Connection con;
String url = "jdbc:mysql://localhost:3306/db_utilisateur";
String user = "mysql";      String pass = "azerty";
try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        con = DriverManager.getConnection(url, user, pass);
} catch (Exception e) {/* … */}
```

Exécuter une requête SQL

```
import java.sql.PreparedStatement;  // entre la déclaration du package et de la classe
//[...]
try { // voir ci-dessus pour se connecter en général par une méthode d'une classe technique
        String rqSql = "INSERT INTO utilisateur(pseudo,nom,prenom) VALUES (?,?,?)";
        PreparedStatement ps = con.prepareStatement(rqSql);
        ps.setString(1,"Amon");      ps.setString(2,"AmontchSky");      ps.setString(3,"Dada");
        ps.executeUpdate();
} catch (Exception e) { /*… */ }
```

Exploiter une requête SELECT

```
import java.sql.PreparedStatement;  // entre la déclaration du package et de la classe
import java.sql.ResultSet;
//[...]
try { // voir ci-dessus pour se connecter en général par une méthode d'une classe technique
        String rqSql = "SELECT * FROM utilisateur";
        PreparedStatement ps = con.prepareStatement(rqSql);
        ResultSet result = ps.executeQuery();
        while (result.next()) {
                System.out.print(result.getString("pseudo") + " "
                        + result.getString("nom") + " "+ result.getString("prenom") + "\n"
                );
        }
} catch (Exception e) { /*… */ }
```

Programmation Orientée Objet

Portée des membres (attributs et méthodes)

Accessible Depuis	La classe	Le package	Classes dérivées	L'extérieur
public	Oui	Oui	Oui	Oui
protected	Oui	Oui	Oui	NON
Par défaut	Oui	Oui	NON	NON
private	Oui	NON	NON	NON

Convention de nommage

Classes : La première lettre de chaque mot (uniquement de noms) composant le nom doit être en majuscule(EcranAbout).

Variables : Première lettre en minuscule puis première lettre de chaque mot composant le nom en majuscule (exemple : utilNom).

Méthodes (Forme verbeNom) : Première lettre en minuscule puis première lettre de chaque mot en majuscule (getUtilNom).

Constantes : Le nom d'une constante est en majuscule. Les mots composant le nom d'une constante doivent être séparés par un caractère de soulignement (exemple : NUM_VERSION).

Nb) Constante : `final int NUM_V = 17; // public et/ou static si besoin.`

Constructeur

```
      // Méthode appelée à la création de l'objet pour l'initialiser
//  [...]
      public class Mere {
            protected String nom;
            public Mere(String nom) {   // Constructeur de la classe mère
                  this.nom = nom
            }
      }
//  [...]
      public final class Fille extends Mere {
            private String numSecu;
            public Fille(String nom, String numSecu) {       // Constructeur de la classe fille
                  super(nom);                                // super() appel le constructeur de la mere
                  this.numSecu = numSecu;
            }
      }
```

Héritage

Ci-dessus, la classe Fille() hérite des propriétés et méthodes de la classe Mere()
le mot-clé **extends** précise que la sous-classe hérite de la super classe
L'héritage d'une classe déclarée **final** est **impossible**

Polymorphismes

Dans une classe Fille(), les méthodes héritées peuvent être redéfinies
(=> même signature de méthode mais comportement différent)

Signature de méthode

Dans une classe, une signature de méthode **doit être unique.**
La signature de méthode est constituée du nom, de la liste et du type des arguments.
Toute modification de cette liste donnera naissance à une nouvelle méthode.
Lorsqu'une méthode est ainsi redéfinie, on parle de **surcharge de méthode.**

Abstraction

```
            // Classe ne pouvant pas être instancié directement
//  [...]
      public abstract class Mere {
            public abstract void methodeA(); // méthode abstraite à définir
            public void methodeB() {
                  System.out.println(this.nom) ;
      }     }
//  [...]
      public class Fille extends Mere {
            public void methodeA() {    // Définition de la méthode abstraire
//  [...]
            }
      }
```

Interface

```
            // Classe ne pouvant être instancié et dont les méthodes sont à definir
//  [...]
      interface class Mere {
            public void methodeA(); // méthode à définir
            public void methodeB() ; // méthode à définir
      }
//  [...]
      public class Fille extends Mere {
            public void methodeA() {    // Définition des méthodes A et B
//  [...]
            }
            public void methodeB() {
                  System.out.println(this.nom) ;
            }
      }
```

Gestion des Exceptions

Gérer les erreurs

Certaines instructions sont peuvent générer des erreurs, Java permet de les gérer :

- les **traiter dans la méthode** par un (**try**, **catch** et **finally**)
- En **déléguer la gestion** à la méthode appelante (**throws Exception**)

try, catch et finally

```
import java.sql.*;
//[...]
Connection con;
String url = "jdbc:mysql://localhost:3306/db_utilisateur",user = "mysql", pass = "azerty";
try {                                   // code à exécuter
        Class.forName("com.mysql.cj.jdbc.Driver");
        con = DriverManager.getConnection(url, user, pass);
} catch (Exception e) {                 // traitement de l'erreur
        System.out.println("Erreur : " + e.getMessage());
} finally {                             // Exécuté après le try catch (finally est optionnel)
        System.out.println("Fin du try catch");
}
```

throws Exception

```
import java.sql.*;
//[...]
public static Connection getConnectDB() throws Exception {
        if (con == null ) {
                Class.forName(driver);
                con = DriverManager.getConnection(url, user, pass);
        }
        return con;
}
//[...]
public static void traitementDB() {
        try {
                Connection con = UtilDB.getConnectDB();
        } catch (Exception e) {
                System.out.println("Erreur : " + e.getMessage());
        }
}
```

JUnit : Test unitaire

Environnement de test

Une classe de test hérite de la classe **TestCase** et implique :
d'instancier les objets utiles, d'appeler des traitements, de vérifier grâce aux méthodes **assert*()** (où ***=Equals, False, True, Same…).

Se connecter

```
import junit.framework.*;
//[...]
public class MaClasseTest extends TestCase {

        // créer les objets et les variables utiles aux tests
//[...]
public void testGetNom() {
        assertEquals("Le nom est incorrect", "nom1", personne.getNom());
}

public void testSetNom() {
        personne.setNom("nom2");
        assertEquals("Le nom est incorrect", "nom2", personne.getNom());
}
```

http://www.jmdoudoux.fr/accueil_java.htm
https://www.w3schools.com/java/default.asp
http://netbeans.apache.org/kb/docs/java/
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html