# Licence 3 MAI Mathématiques Appliquées à l'info (MAI)

## Développement mobile

Séquence 1: Initiation à la programmation Orientée Objet

Mr. Boubou CAMARA

# Plan

## Séquence 1 : Initiation à la programmation Orientée Objet

**0. Préambule**

**1.Fondements de la programmation OO**

**1.1 Objet, Classe, Interface, Package**

**1.2 Encapsulation, Message**

**1.3 Héritage, Surcharge, redéfinition**

**2. Fondements du langage Java**

**2.1 JVM, JDK, JRE, compilation et interprétation**

**2.2 Structure de controle**

**2.3 Exception et Finally**

**3. Programmation Java avancées**

**3.1 Gestion de fichiers**

**3.2 Communication réseau**

**3.3 Multithreading**

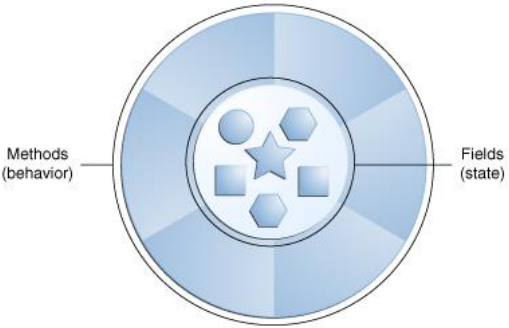
# Préambule - Programmation Orientée Objet

La programmation orientée objet (OOP) est un paradigme de programmation basé sur le concept d'objets. Un objet peut contenir des données, sous la forme de champs, souvent appelés attributs; et le code, sous la forme de procédures, souvent appelées méthodes. Les objets communique entre eux par l'envoi de messages. La programmation OO simplifie le développement et la maintenance de logiciels en se basant sur quelques concepts:

- Objet
- Classe
- Héritage
- Polymorphisme
- Abstraction
- Encapsulation

Il existe plusieurs langages Orientés Objets : Smalltalk, Simula, C#, Java…..
Dans ce cours nous nous intéresserons au langage Java.

# Préambule - Programmation Orientée Objet (POO)



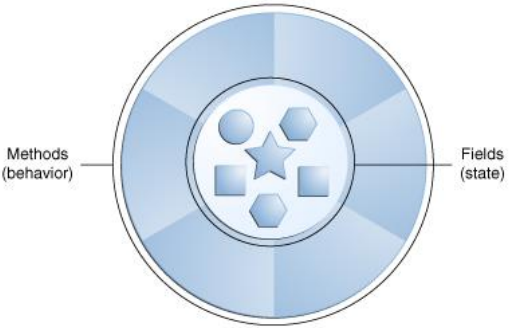

Les objets sont essentiels à la compréhension de la technologie orientée objet. Il y'a de nombreux exemples d'objets du monde réel: le téléphone, le bureau, la télé.

« classe »

Une classe est un modèle ou un prototype à partir duquel des objets sont créés. Ces objets partagent le même ensemble d'attributs et d'opérations. Exemple de déclaration de classe en Java

```
public class Personne {
    private int age;
    private String nom;
    public int getAge() {
        return age;
    }
    public boolean getAge() {
        return age;
    }
    public void setAge(int age){
        this.age=age;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom){
        this.nom=nom;
    }}
```

# Préambule - Programmation Orientée Objet (POO)




«objet »

Un «objet» est une représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés (attributs ou membres) et des actions (méthodes). C'est une instance d'une classe. Exemple de création d'un objet Personne en Java.

```
Personne fatou = new Personne();
fatou.setAge(23);
tatou.setNom("Fatou");
```

«attributs»

Les «attributs» (aussi appelés «données membres») sont les caractères propres un «objet». Exemple: l'«objet» personne a les «attributs» suivants: nom, age, prenom, taille, sexe, dateDeNaissance, etc ..

«méthode »

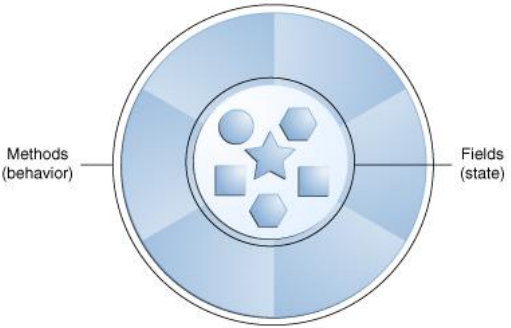
Les «méthodes » sont les actions applicables à un «objet», ou encore les messages compréhensibles par cet «objet». Dans notre exemple, la classe «Personne » a les méthodes suivantes: setAge, getAge, marche, mange, estMajeur, estUnHomme, estMineur,

«encapsulation »

c'est un concept important en POO, qui signifie regrouper dans une seule entité les attributs d'un objet ainsi que ses méthodes. On ne peut manipuler/accéder directement les propriétés d'un objet, on passe par les méthodes. Exemple : dans l'exemple précédent si on veut modifier l'âge de l'objet **fatou** , on doit passer par la méthode setAge:

```
fatou.setAge(25);
```

# Préambule - Programmation Orientée Objet (POO)




«signature d'une méthode »
la signature d'une méthode c'est le nom de la méthode et sa liste de paramètres ainsi que leurs types. Exemple: **setAge(int age)** est la signature de la méthode setAge de notre classe Personne.

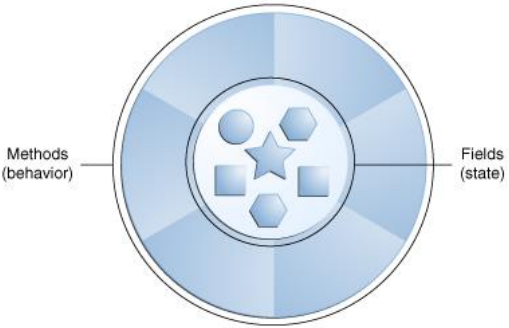
«instanciation »
c'est quand on construit un objet à partir de sa classe. On dit que cet objet est une instance de la classe. On dit aussi qu'on instancie la classe. Exemple: fatou est une instance de la classe Personne.

«classe abstraite»
Une classe abstraite est une classe qui est déclarée abstraite; elle peut ou non inclure des méthodes abstraites. Les classes abstraites ne peuvent pas être instanciées, mais elles peuvent être sous-classées. Exemple:

```
public abstract class Forme {
    int x,y;
    Form(int x, int y) {
        this.x=x;
        this.y=y;
    }
    abstract double perimetre();
    abstract double surface();

    public void setNom(String nom){
        this.nom=nom;
    }
}
```

# Préambule - Programmation Orientée Objet (POO)




«interface »

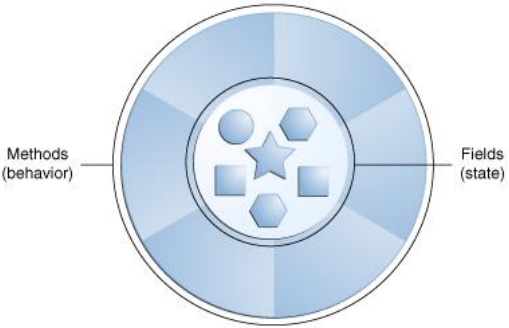
Une interface est un contrat entre une classe et le monde extérieur. Elle déclare sans définir leur contenu, l'ensemble de ses méthodes et ses constantes. Exemple d'interface en java:

```
public interface MonInterface {
    int constant = 20; // constante
    int maMethod(); // signature de methode
}
```

héritage

L'héritage fournit un mécanisme pour organiser et structurer les classes. Une classe qui dérive d'une autre classe s'appelle une sous-classe (également une classe dérivée, une classe étendue ou une classe enfant). La classe à partir de laquelle la sous-classe est dérivée s'appelle une superclasse (également une classe de base ou une classe parent).

# Préambule - Programmation Orientée Objet (POO)




```
public class Rectangle extends Forme {
    private final double largeur, longueurs; //cotés
    public Rectangle(){this(1,1);}
    public Rectangle(double largeur, double longueur) {
        this.largeur = largeur;
        this.longueur = longueur;
    }
    @Override
    public double aire() {
        // A = l * L
        return largeur * longueur;
    }

    @Override
    public double perimetre() {
        // P = 2(largeur + longueur)
        return 2 * (largeur + longueur);
    }
}
```

# Préambule - Programmation Orientée Objet

notion de « package »

Un «package» est un espace de noms pour organiser des classes et des interfaces de manière logique. Placer votre code en paquets rend les projets logiciels plus faciles à gérer. Exemple de packages proposés dans le kit de développement java (JDK, nous y reviendrons) et de classes qu'ils contiennent:

| package | Exemples de classes | Exemple de classes/interfaces |
|---|---|---|
| **java.lang** package de base de java; ce package est importé implicitement dans toute classe | **Objet**<br>**Exception**<br>**Integer**<br>**String** | superclasse de toutes les classes<br>classe de bas des exceptions<br>classe enveloppe du type primitif int<br>chaines de caractères non modifiables |
| **java.util** Utilitaires diverses | **List**<br>**Set**<br>**Collections** | interface des tableaux extensibles<br>interface des collections sans doublons<br>classe utilitaires retournant des collections |
| **java.net** utiliser des fonctionnalités réseau | **Url**<br>**Authenticator** | classe représentant une URL<br>classe qui sait comment obtenir une l'authentification d'une connexion réseau |
| **java.math** Utiliser des opérations mathématiques | **BigDecimal** | classe permettant de réaliser des calculs en virgule flottante avec précision |

# Fondements du langage Java - définitions

1.Qu'est-ce que Java?
Le terme Java désigne à la fois un langage de programmation, une machine virtuelle (JVM : Java Virtuel Machine) et une plateforme (la plateforme Java).
1.1 Le langage Java est un langage de programmation « orienté Objet » utilisé pour écrire des applications java (qui peuvent être de types : «autonomes», «applications web», «applications d'entreprise», «applications mobiles»)

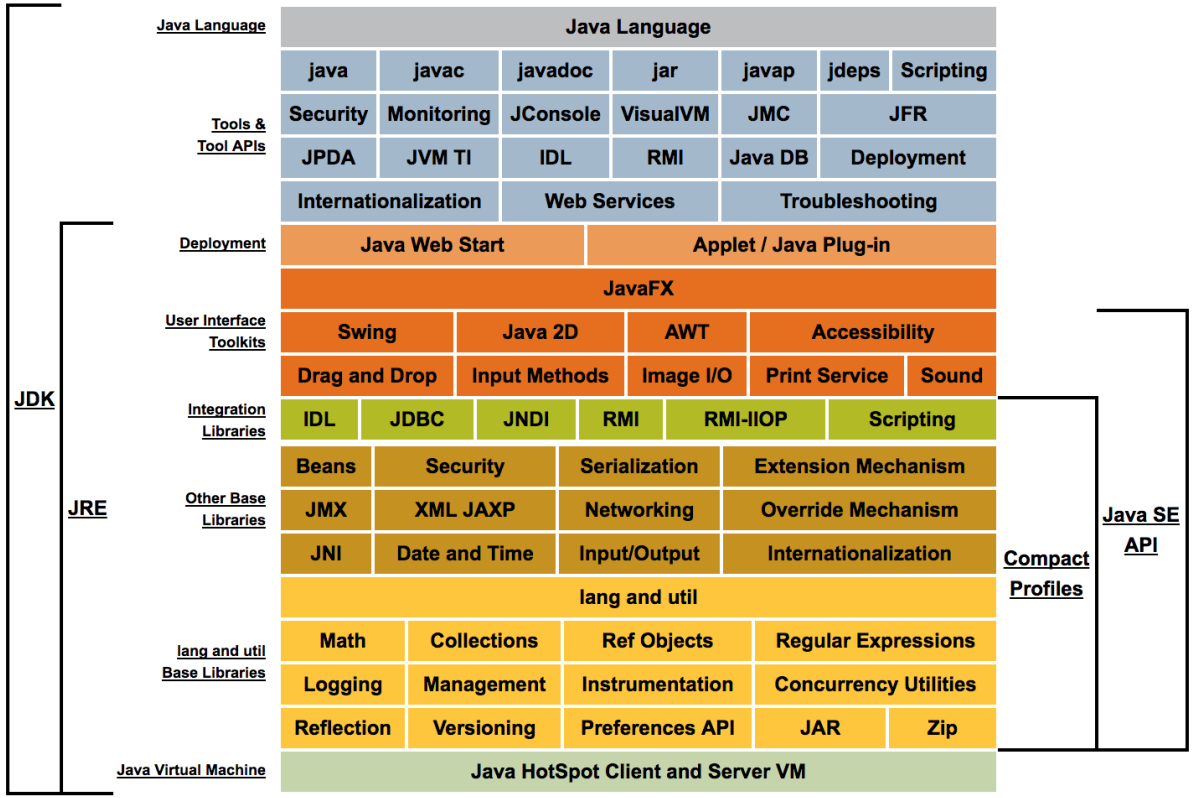

Le/les fichiers de classe/interface sont écrits dans le langage Java, avec une extension **.java;** une fois compilés avec le compilateur java, des fichiers de code byte code, d'extension **.class** sont générés. Ce code compilé peut-être exécuté par n'importe quelle JVM répondant aux spécifications de SUN (Oracle).

# Fondements du langage Java - définitions

1.2 La machine virtuelle Java JVM (Java Virtual Machine) est une machine abstraite. C'est une spécification qui fournit un environnement d'exécution dans lequel le bytecode java (format .class, résultant de la compilation d'une classe java) peut être exécuté. La JVM effectue les tâches suivantes:

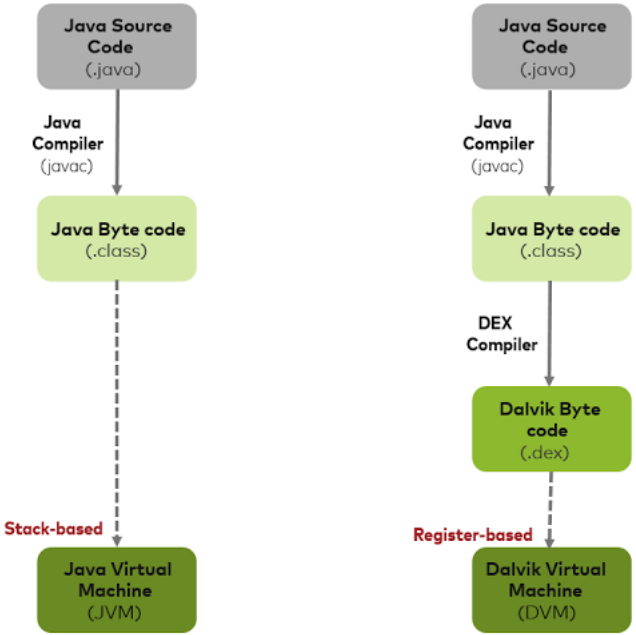
* Charge le byte code
* Vérifie le code
* Exécute le code
* Fournit un environnement d'exécution

# Fondements du langage Java - définitions




Le schéma à gauche montre les éléments du langage Java, de la plateforme et de la JVM de Sun Microsystems (maintenant Oracle)

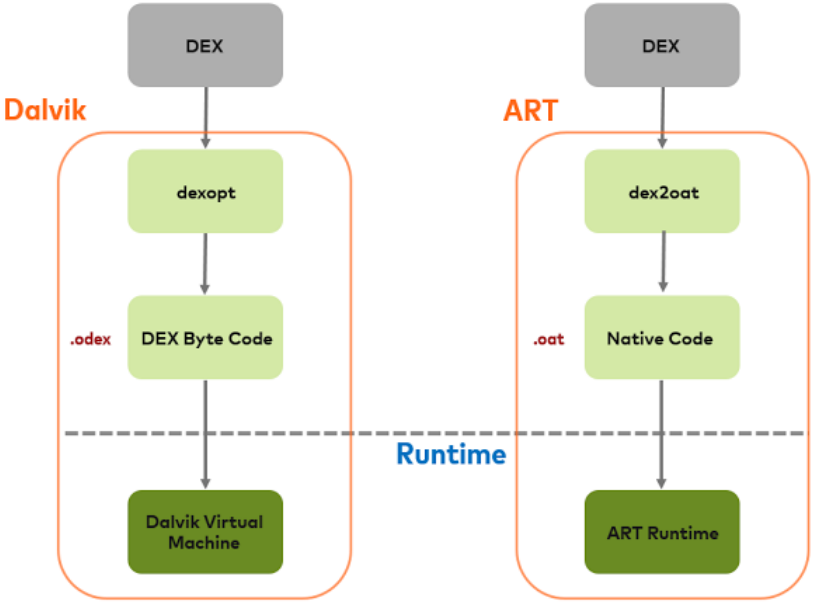
# Fondements du langage Java pour Android : JVM vs DVM




JVM vs DVM

Même si le langage de programmation utilisé pour développement d'application mobile pour Android est Java, il y'a quelques différences au niveau du byte code généré et de l'environnement d'exécution. Contrairement à une application classique java, dont le byte code (.class) est exécuté sur la JVM, pour Android le byte code subit une deuxième compilation par le compilateur Dalvik qui génère un fichier **.dex**. C'est ce dernier qui est exécuté par la machine virtuelle Dalvik (DVM).

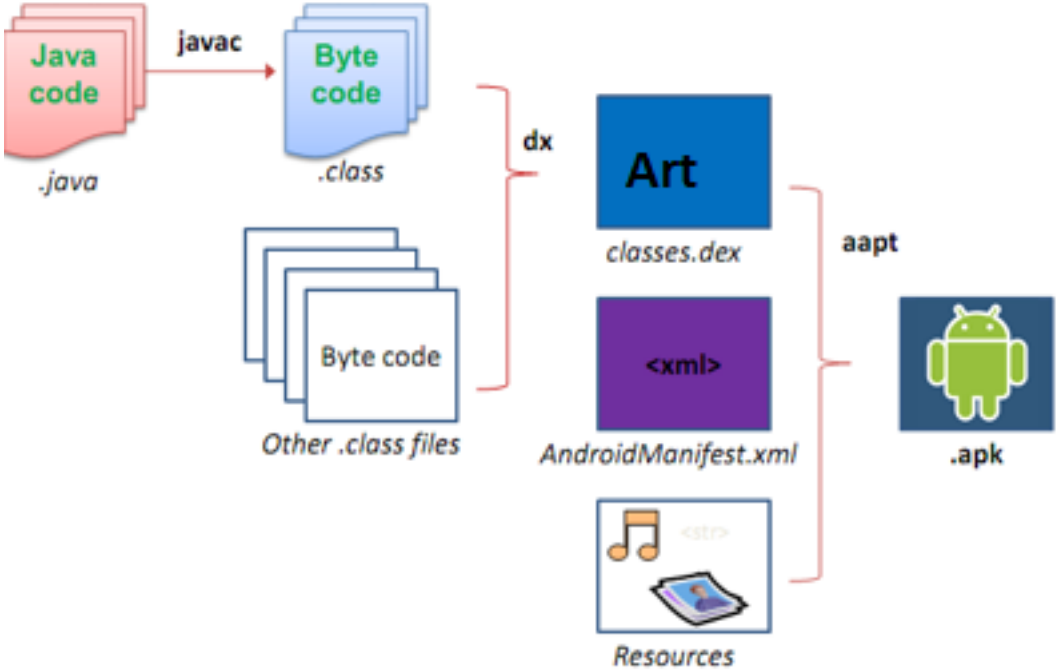
# Fondements du langage Java - définitions




Dalvik vs ART

Depuis la version 5.0 (Lollitop) d'Android, la machine virtuelle Dalvik(DVM) a été remplacée par la machine virtuelle Android (ART) plus performante.

# Java Android : première application




1. Android et Java, qu'est-ce qui lie les deux?
   (a) l'API Android : c'est un ensemble de routines, des protocoles, des classes qui jouent un rôle important dans la création des applications Android. Une partie de l'API Java est incluse dans Android.
   (b) Une fois le code java développé en utilisant les librairies de l'API Android, il est compilé et placé dans un fichier apk (Android package): une collection de fichiers compressés pour le système d'exploitation Android.
2. Mise en place de l'environnement de développement
   (a) installer le JDK de Oracle
   (b) installer Android Studio

# Java Android : Prérequis

Il n' y a pas de limitations sur les S.E sur la machine du développeur d'application native Android.
Pour installer Android Studio les pré-requis suivants doivent être respectées par le développeur doit:

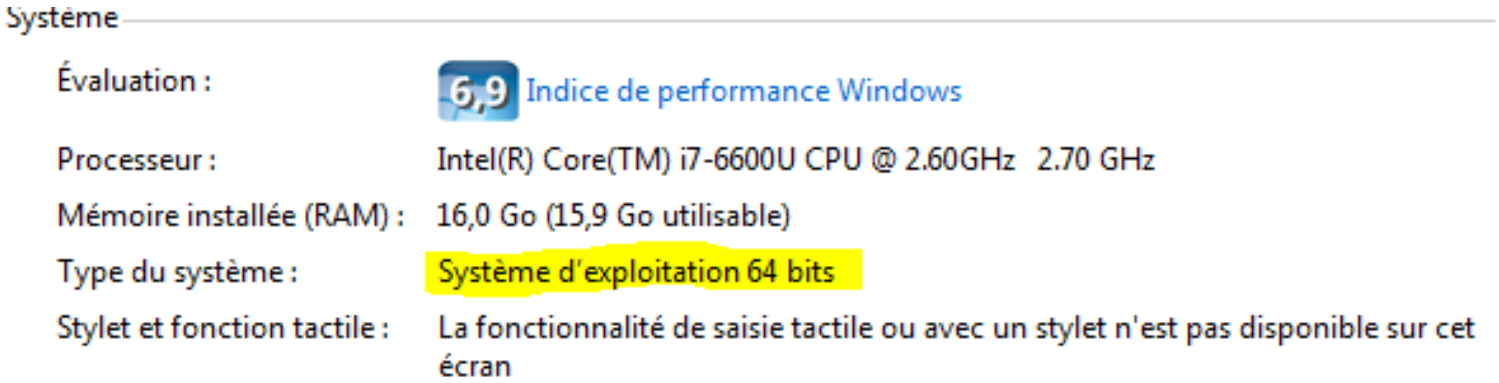
1. disposer d'un ordinateur avec un minimum de 2Go de RAM et 1,5 Go d'espace disque
2. avoir un compte administrateur de son ordinateur
3. installer le JDK (Kit de développement Java)
4. installer :
    1. Android Studio, l'EDI standard de développement Android
5. Une fois l'application développée et testée dans le Simulateur, on peut le tester sur un appareil physique Android. Par contre, pour le déployer sur GooglePlay, il doit :

3.1 s'inscrire au programme pour développeur Android de Google pour 25 $ à vie
3.2 détenir un appareil Android

# Java Android : installation du JDK

Dans cette section nous allons décrire le processus d'installation du JDK, si il n'est pas déjà fait;

1. installation sur SE Windows
   (a) determination 32bits vs 64bits :
      (a) Sous Windows Vista/7 : Cliquez sur le bouton Démarrer > Panneau de configuration > Système et sécurité > Système
      Sous l'aperçu on devrait voir d'écran ci-dessous. Si rien n'est écrit, vous êtes très certainement en 32 bits




   (b) Windows - téléchargement, installation et configuration du JDK
      (i) visiter la page de téléchargement du JDK à : http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html

# Java Android : installation du JDK - Windows

(b) Windows - téléchargement, installation et configuration du JDK

(i) visiter la page de téléchargement du JDK à : http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html

(ii) les fenêtre suivante s'affiche, il faudrait tout d'abord « Sélectionner « Accepter le contrat de Licence » et choisir la version qui convient: Windows x86 pour du 32bits, ou Windows x64 pour du 64 bits.

**Java SE Development Kit 8u141**

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement  Decline License Agreement

| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 32 Hard Float ABI | 77.88 MB | jdk-8u141-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 74.83 MB | jdk-8u141-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 164.66 MB | jdk-8u141-linux-i586.rpm |
| Linux x86 | 179.4 MB | jdk-8u141-linux-i586.tar.gz |
| Linux x64 | 162.11 MB | jdk-8u141-linux-x64.rpm |
| Linux x64 | 176.92 MB | jdk-8u141-linux-x64.tar.gz |
| Mac OS X | 226.6 MB | jdk-8u141-macosx-x64.dmg |
| Solaris SPARC 64-bit | 139.84 MB | jdk-8u141-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 99.17 MB | jdk-8u141-solaris-sparcv9.tar.gz |
| Solaris x64 | 140.59 MB | jdk-8u141-solaris-x64.tar.Z |
| Solaris x64 | 97.01 MB | jdk-8u141-solaris-x64.tar.gz |
| Windows x86 | 190.95 MB | jdk-8u141-windows-i586.exe |
| Windows x64 | 197.78 MB | jdk-8u141-windows-x64.exe |

Back to top

(iii) effectuer le téléchargement; une fois le téléchargement terminé, sélectionner le fichier téléchargé, et « Exécuter le fichier en tant qu'administrateur ». Choisir un répertoire d'installation: C:\java\jdk

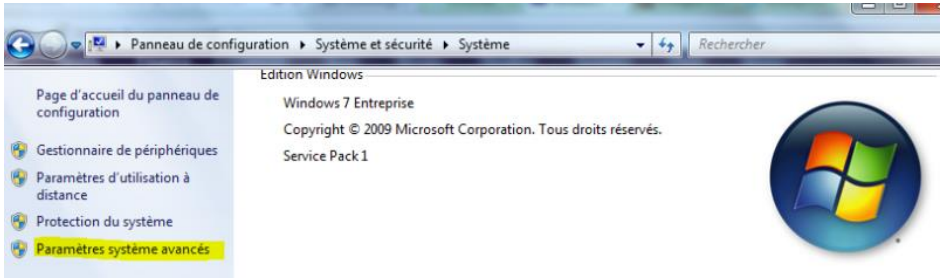
# Java Android : installation du JDK - Windows

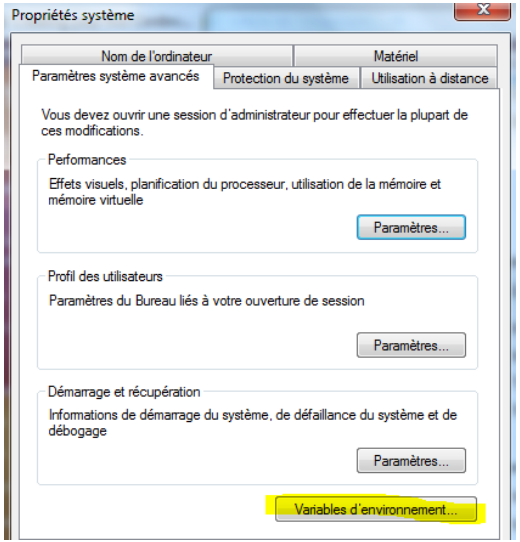
(b) Windows - téléchargement, installation et configuration du JDK

(iv) une fois l'installation du jdk et du JRE sur le répertoire C:\java\jdk terminé, nous allons nous assurer que Windows va pouvoir retrouver le JDK. Pour cela, on va :

- créer une variable d'environnement JAVA_HOME
- faire pointer cette variable vers le chemin ou le JDK a été installé : pour notre cas **C:\java\jdk**

**CREATION DE LA VARIABLE D'ENVIRONNEMENT**



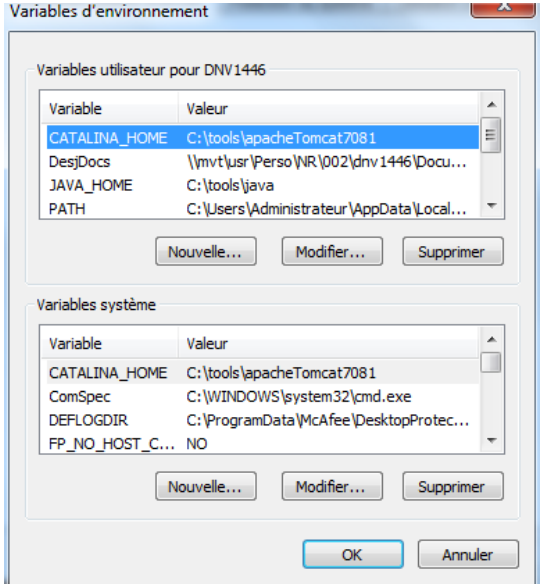
# Java Android : installation du JDK - Windows

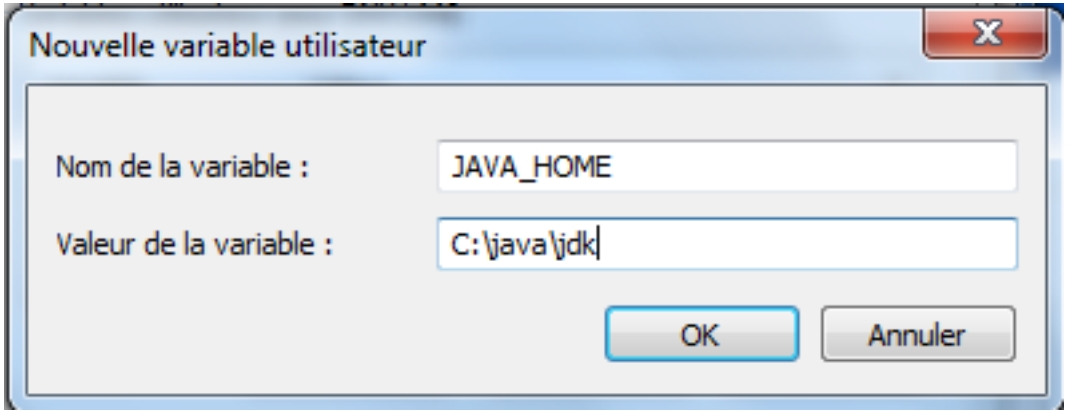
(b) Windows - téléchargement, installation et configuration du JDK

(iv) une fois l'installation du jdk et du JRE sur le répertoire C:\java\jdk terminé, nous allons nous assurer que Windows va pouvoir retrouver le JDK. Pour cela, on va :

- créer une variable d'environnement JAVA_HOME
- faire pointer cette variable vers le chemin ou le JDK a été installé : pour notre cas **C:\java\jdk**

**CREATION DE LA VARIABLE D'ENVIRONNEMENT JAVA_HOME**



# Java Android : installation du JDK - MacOS

2. installation sur MacOS
   (i) visiter la page de téléchargement du JDK à : [http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html](http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html)
   (ii) les fenêtre suivante s'affiche, il faudrait tout d'abord « Sélectionner « Accepter le contrat de Licence » et choisir la version qui convient: Mac OSX.

**Java SE Development Kit 8u141**

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement  Decline License Agreement

| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 32 Hard Float ABI | 77.88 MB | jdk-8u141-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 74.83 MB | jdk-8u141-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 164.66 MB | jdk-8u141-linux-i586.rpm |
| Linux x86 | 179.4 MB | jdk-8u141-linux-i586.tar.gz |
| Linux x64 | 162.11 MB | jdk-8u141-linux-x64.rpm |
| Linux x64 | 176.92 MB | jdk-8u141-linux-x64.tar.gz |
| Mac OS X | 226.6 MB | jdk-8u141-macosx-x64.dmg |
| Solaris SPARC 64-bit | 139.84 MB | jdk-8u141-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 99.17 MB | jdk-8u141-solaris-sparcv9.tar.gz |
| Solaris x64 | 140.59 MB | jdk-8u141-solaris-x64.tar.Z |
| Solaris x64 | 97.01 MB | jdk-8u141-solaris-x64.tar.gz |
| Windows x86 | 190.95 MB | jdk-8u141-windows-i586.exe |
| Windows x64 | 197.78 MB | jdk-8u141-windows-x64.exe |

Back to top

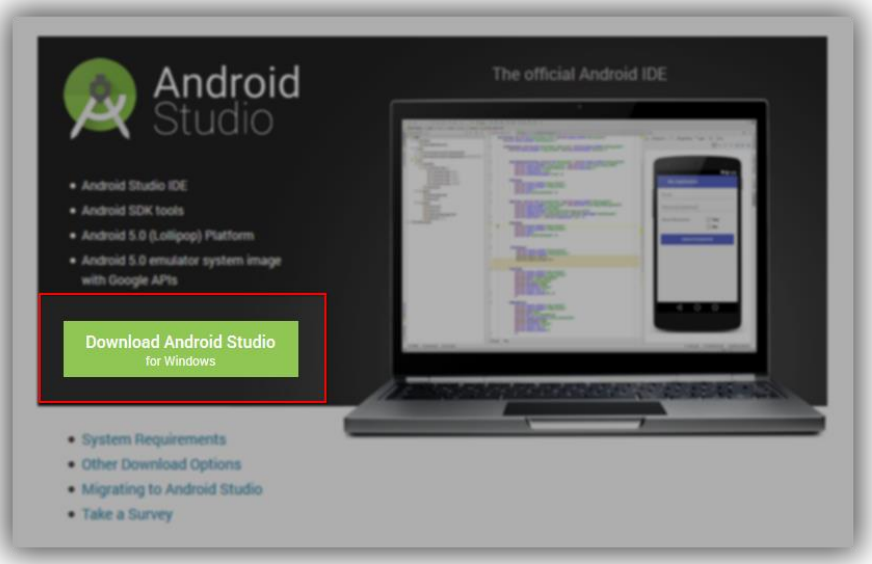
(iii) effectuer le téléchargement; une fois le téléchargement terminé, sélectionner le fichier téléchargé, et l'exécuter.

# Java Android : installation du JDK - MacOS

2. installation sur MacOS

   (iv) garder pour l'installation les répertoires par défaut proposés par le système.

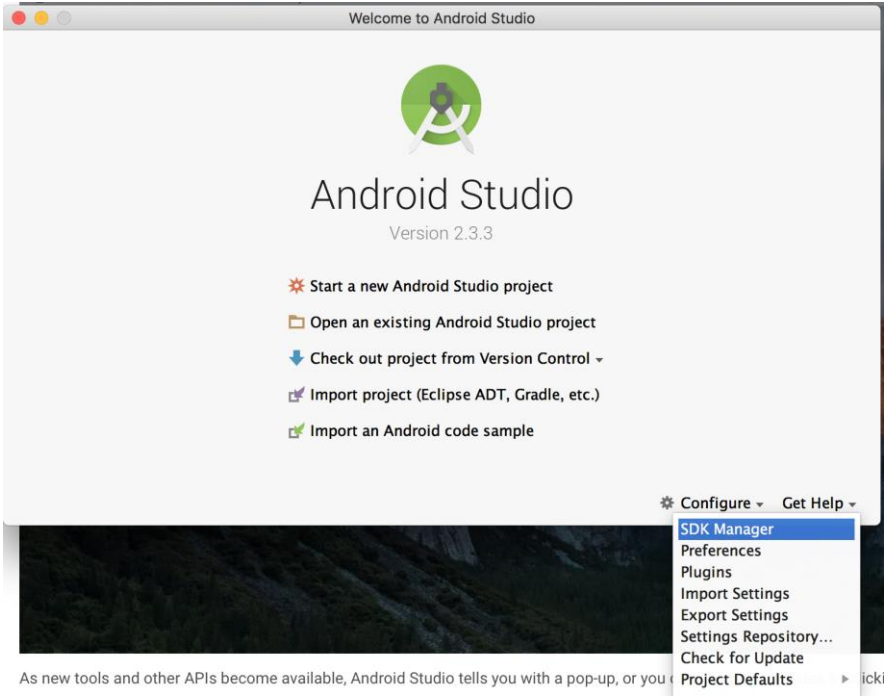
# Java Android- Installation de Android Studio




**Téléchargement et installation de Android Studio et Android SDK**

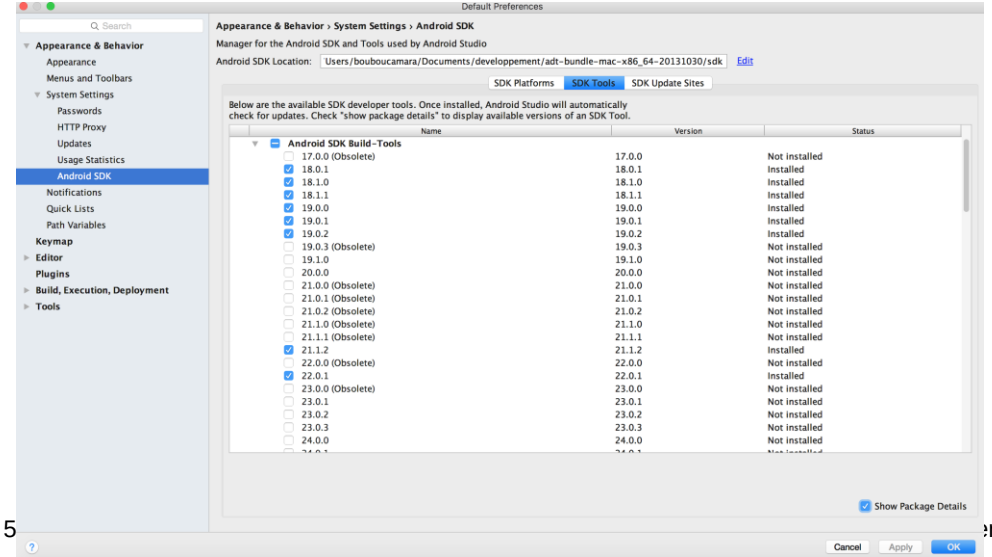
1. Se rendre sur le site https://developer.android.com/sdk/installing/studio.html pour télécharger un paquet qui contient à la fois l'EDI Android Studio et un outil pour gérer l'installation du SDK Android sur votre système.
2. Une fois le téléchargement terminé, compléter l'installation en suivant les instructions.
3. La fenêtre suivante s'ouvre

# II.3.c Android - Comment ça marche - Installation de Android Studio




**Téléchargement et installation de Android Studio et Android SDK**

4. cliquer sur SDK Manager, une fenêtre similaire à la suivante s'affiche



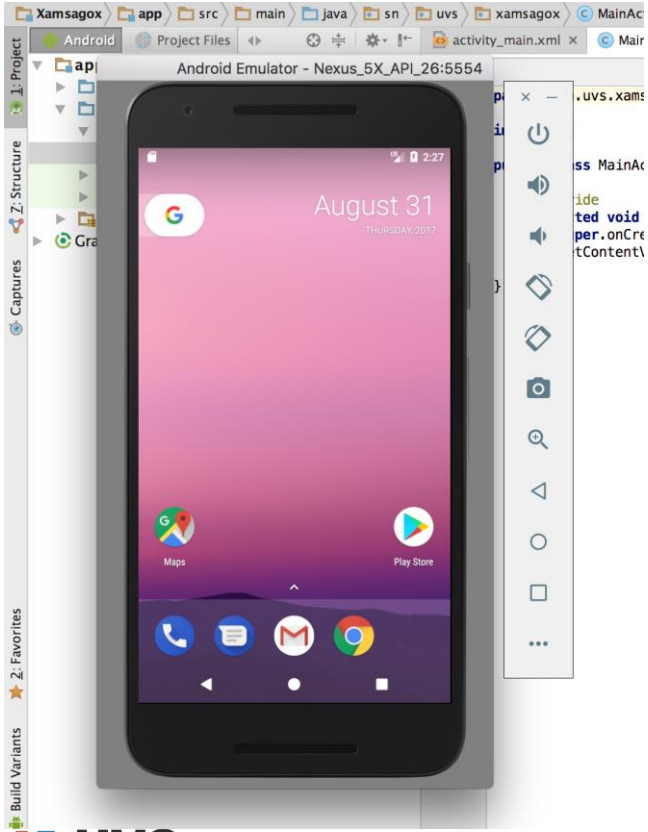

5 [illegible] sur le parc d'utilisateurs visé. Le site internet suivant, https://developer.android.com/about/dashboards/index.html, fournit la proportion d'appareil par version de la plateforme Android. En date du 30 Août 2017, c'est la version 6.0 de la plateforme, dénommée **Marshmallow**, qui est la plus distribuée avec 32,3%.

6. une fois l'installation terminée , il est temps d'explorer Android Studio et développer votre première application Android

# II.3.c Android - Comment ça marche - Création d'un projet Android Studio


Android Studio comprend:
• un concepteur GUI
• un éditeur de code avec support pour la coloration syntaxique et la numérotation de ligne
• auto-retrait et auto-complétion (c.-à-d. Type d'indexation)
• un déboggueur
• un système de contrôle de version
• un support de refactorisation du code
et plus.

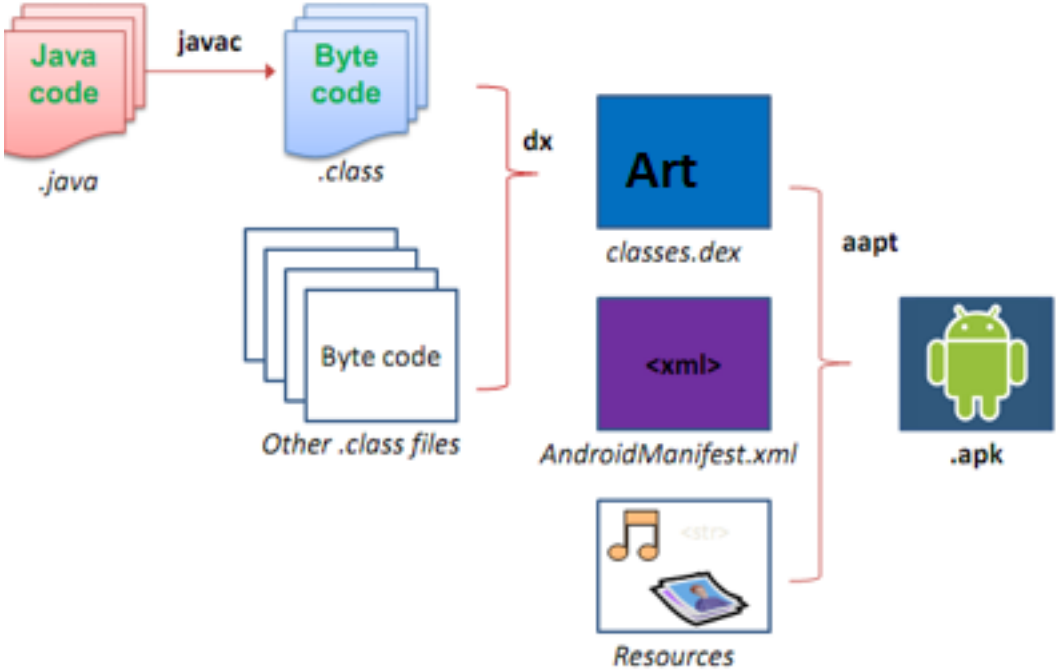
# II.3.e Android - Utilisation de l'émulateur




L'émulateur Android, inclus dans le SDK d'Android, vous permet d'exécuter des applications Android dans un environnement simulé sous Windows, Mac OS X ou Linux, sans utiliser de Appareil Android. L'émulateur affiche une fenêtre d'interface utilisateur Android réaliste. C'est particulièrement utile si vous n'avez pas accès aux appareils Android pour les tests. Vous devriez certainement tester vos applications sur une variété d'appareils Android avant de les télécharger sur Google Play.

Avant d'exécuter une application dans l'émulateur, vous devrez créer un Android Virtual Dispositif (AVD), qui définit les caractéristiques de l'appareil sur lequel vous souhaitez tester, y compris le matériel, l'image du système, la taille de l'écran, le stockage des données et plus encore.

# Java Android - installation de Android Studio




1. Android et Java, qu'est-ce qui lie les deux?
   (a) l'API Android : c'est un ensemble de routines, des protocoles, des classes qui jouent un rôle important dans la création des applications Android. Une partie de l'API Java est incluse dans Android.
   (b) Une fois le code java développé en utilisant les librairies de l'API Android, il est compilé et placé dans un fichier apk (Android package): une collection de fichiers compressés pour le système d'exploitation Android.

# Java - Structure de controle

Les instructions de contrôle de flots permettent d'interrompre le flux d'exécution en utilisant la prise de décision, le bouclage et le branchement.

1. instructions de prise de décision
   **(a) if-then**
   **(b) if-then-else**

2. instructions de bouclage
   **(a) for**
   **(b) while**
   **(c) do while**

3. instructions de branchement
   **(a) for**
   **(b) while**
   **(c) do while**

# Java - Gestion des exceptions

Le langage de programmation Java utilise des exceptions pour gérer les erreurs et autres événements exceptionnels. Cette leçon décrit quand et comment utiliser les exceptions.

Une exception est un événement qui se produit lors de l'exécution d'un programme qui perturbe le flux normal d'instructions.

Lors de la détection d'une erreur, un objet qui hérite de la classe Exception est créé (on dit qu'une exception est levée) et propagé à travers la pile d'exécution jusqu'à ce qu'il soit traité.

Quand on écrit une classe qui utilise le code avec une possibilité d'échec, on peut préparer les utilisateurs de la classe en utilisant des exceptions avec **try**, **catch** et **finally**.

Nous pouvons écrire des méthodes dans nos classes en utilisant le mot-clé Java **throws** à la fin de la Signature de la façon suivante:

```
public void uneMethodeRisquee () throws UneException {
// Le code risqué va ici
}
```

Une code qui utilise **uneMethodeRisquee** devra gérer l'exception. La gestion des exceptions consiste à envelopper le code dans les blocs **try** and **catch** et **finally** :

# Java - Gestion des exceptions

```
try {
  uneMethodeRisquee();
} catch (MonException me) {
      //traitement pour MonException
} finally {
      // traitement qui sera toujours exécuté
}
```

En option, la clause **finally** définit un bloc qui sera toujours exécuté, qu'une exception soit levée ou non.

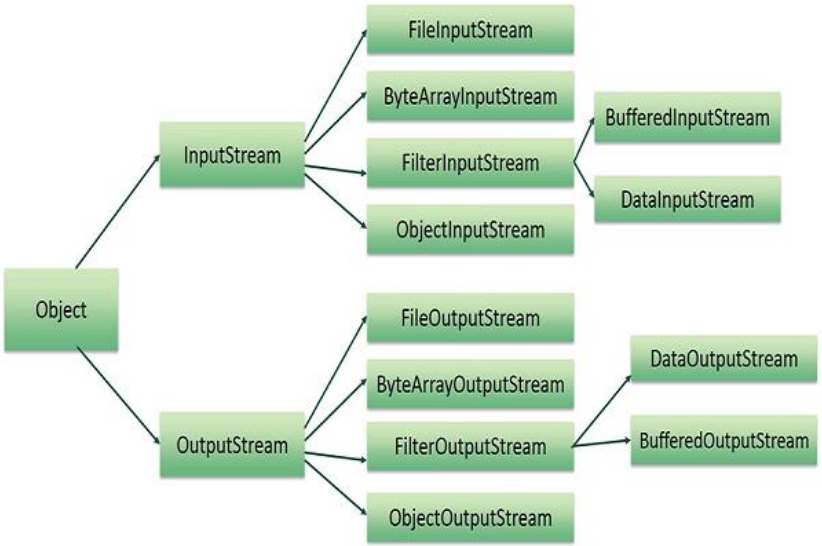
# Programmation avancée Java - Gestion de fichiers

Une entrée/sortie en Java consiste en un échange de données entre le programme et une autre source, par exemple la mémoire, un fichier, etc. Le paquet java.io contient les classes pour effectuer des entrées et des sorties (E / S) en Java

Java emploie les flux (**stream** ) entre la source et la destination des ipes:

Ouverture d'un flux:

| | Lecture | Ecriture |
|---|---|---|
| **Texte** | Reader | Writer |
| **Binaire** | InputStream | OutputStream |




4 classes abstraites principales avec des opérations :

- de lecture/écriture sur des données de type binaire/textuelles

hiérarchie entre les classes traitant les données binaires

# Programmation avancée Java - Gestion de fichiers

## Lecture et écriture de fichiers

Les deux flux importants sont FileInputStream et FileOutputStream, qui seront abordés dans ce didacticiel.

## FileInputStream

Ce flux est utilisé pour lire des données à partir des fichiers. Les objets peuvent être créés en utilisant le mot-clé **new** et il existe plusieurs types de constructeurs disponibles. Le constructeur suivant prend un nom de fichier en tant que chaîne pour créer un objet de flux de saisie pour lire le fichier: **InputStream f = new FileInputStream ("C:/java/hello »);**

| | | |
|---|---|---|
| **public void close() throws IOException{}**<br>Cette méthode ferme le flux de sortie du fichier. Libère toutes les ressources système associées au fichier. lance une exception de type IOException. | **public int read(int r)throws IOException{}**<br>Cette méthode lit l'octet spécifié des données de InputStream. Renvoie un int. Renvoie le prochain octet de données et -1 sera retourné si c'est la fin du fichier. | **public int available() throws IOException{}**<br>Gives the number of bytes that can be read from this file input stream. Returns an int. |
| **protected void finalize()throws IOException {}**<br>cette méthode nettoie la connexion au fichier. Assure que la méthode de fermeture de ce flux de sortie de fichier est appelée lorsqu'il n'y a plus de références à ce flux. Génère une exception de type IOException. | **public int read(byte[] r) throws IOException{ }**<br>This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned. | |

d'autres opérations sur le flux.

# Programmation avancée Java - Gestion de fichiers

**FileOutputStream**

FileOutputStream est utilisé pour créer un fichier et y écrire des données. Le flux créera un fichier, s'il n'existe pas déjà, avant de l'ouvrir pour la sortie. Les deux constructeurs suivants peuvent être utilisés pour créer un objet FileOutputStream.

1) Le constructeur suivant prend un nom de fichier en tant que chaîne pour créer un objet de flux de saisie pour écrire le fichier:

**OutputStream f = new FileOutputStream ("C:/java/hello")**

2) Le constructeur suivant prend un objet de fichier pour créer un objet de flux de sortie pour écrire le fichier.

| | |
|---|---|
| **public void close() throws IOException{}**<br>Cette méthode ferme le flux de sortie du fichier. Libère toutes les ressources système associées au fichier. lance une exception de type IOException. | **public void write(int w)throws IOException{}**<br>Cette méthode écrit l'octet spécifié dans le flux de sortie. |
| **protected void finalize()throws IOException {}**<br>cette méthode nettoie la connexion au fichier. Assure que la méthode de fermeture de ce flux de sortie de fichier est appelée lorsqu'il n'y a plus de références à ce flux. Génère une exception de type IOException. | **public void write(byte[] w)**<br>Écrit w.length octets du tableau d'octets mentionné dans OutputStream. |

uvent être utilisées pour écrire ou diffuser ou effectuer d'autres opérations sur le flux.

# Programmation avancée Java - Gestion de fichiers

**Exemple d'application de gestion de fichiers**

```
import java.io.InputStream;
import java.io.OutputStream;
public class Demo {
	public static void main(String args[]) {
		try {
			byte[] bWrite = {65,66,67,68,69};
			OutputStream os = new FileOutputStream("test.txt");
			for(int x = 0; x < bWrite.length ; x++) {
				os.write( new Integer(bWrite[x]) );   // ecrire les bytes
				os.flush();
			}
			os.close();
			InputStream is = new FileInputStream("test.txt");
			int size = is.available();
			for(int i = 0; i < size; i++) {
				System.out.print((char)is.read() + "  ");
			}
			is.close();
		} catch(IOException e) {
			System.out.print("Exception");
	}
}
```

L'application crée un fichier test.txt dont le contenu est : ABCDE.

# Programmation avancée Java - Multithreading

**Définitions :**

Un **thread** désigne un « fil d'exécution » dans un programme ; c'est-à-dire une suite linéaire et continue d'instructions qui sont exécutées séquentiellement les unes après les autres. En fait, le langage **Java** est multi-**thread**, c'est-à-dire qu'il peut faire cohabiter plusieurs fils d'exécution de façon indépendante.

Le mot multithreading peut être défini par plusieurs threads de contrôle ou plusieurs flux de contrôle. Alors qu'un processus traditionnel a toujours contenu et contient toujours un seul thread de contrôle, le multithreading (MT) sépare un processus en plusieurs threads d'exécution, chacun fonctionnant indépendamment.

Avantage du Multithreading:

- Améliorer la réactivité des applications

# Programmation avancée Java - Multithreading

Pour obtenir des applications multithread, on démarre, en général avec une méthode main, un premier thread qui peut en lancer d'autres sans s'interrompre pour autant ; les nouveaux threads peuvent à leur tour démarrer d'autres threads. Un thread est quelquefois aussi appelé processus léger ou file d'exécution. Un programme multi-thread, est aussi dit concurrent.
En java, pour créer un thread, on peut soit étendre la classe Thread, ou utiliser l'interface Runnable.
Les points les plus importants pour les thread sont les suivant:

- Chaque Thread a une priorité. Les threads avec une priorité plus élevée sont exécutés avant ceux avec moins de priorité
- Chaque thread peut ou non être marqué comme un démon.
- Il existe deux façons de créer un nouveau thread d'exécution. L'une consiste à déclarer qu'une classe est une sous-classe de Thread;  L'autre façon de créer un thread consiste à déclarer une classe qui implémente l'interface Runnable.
- La classe Thread fournit des constructeurs et des méthodes pour créer et exécuter des opérations sur une classe de threads. La classe Thread étend la classe Object et implémente l'interface Runnable.

# Programmation avancée Java - Multithreading

Constructeurs de la classe Thread

1. Thread ()
2. Thread (String str)
3. Thread (Runnable r)
4. Thread (Runnable r, String str)

La classe Thread définit plusieurs méthodes de gestion des threads dont:

Description de la méthode

**setName ()** pour donner un nom à thread

**getName ()** retourne le nom du thread

**getPriority ()** retourne la priorité du thread

**isAlive ()** vérifie si le thread est toujours en cours d'exécution ou non

**join ()** Attendre qu'un thread se termine

**run ()** Point d'entrée pour un thread

**sleep ()** suspendre le fil pendant un certain temps

**start ()** lance un thread en appelant run () metho

# Programmation avancée Java - Réseau

Le terme de programmation réseau se réfère à l'écriture de programmes qui s'exécutent sur plusieurs périphériques (ordinateurs), dans lesquels les périphériques sont tous connectés les uns aux autres à l'aide d'un réseau.

Le package java.net des API Java Standard contient une collection de classes et d'interfaces qui fournissent les détails de communication de bas niveau, permettant d'écrire des programmes qui se concentrent sur la résolution du problème.

1) Java Socket
    (a) définition : Une socket est un point d'extrémité d'une liaison de communication bidirectionnelle entre deux programmes exécutés sur le réseau.Une Socket est utilisée pour la communication entre les applications s'exécutant sur différents JRE. Les classes de socket sont utilisées pour représenter la connexion entre un programme client et un programme serveur. Le paquet java.net fournit deux classes - Socket et ServerSocket - qui implémentent le côté client de la connexion et le côté serveur de la connexion, respectivement.
    (b) La programmation Java Socket peut être orientée connexion ou sans connexion.
    (c) Les classes **Socket** et **ServerSocket** sont utilisées pour la programmation de socket orientée connexion
    (d) et les classes **DatagramSocket** et **DatagramPacket** sont utilisées pour la programmation de socket sans connexion.
    (e) Le client dans la programmation socket doit connaître deux informations : l'adresse IP du serveur et son numéro de port

# Programmation avancée Java - Réseau

Le terme de programmation réseau se réfère à l'écriture de programmes qui s'exécutent sur plusieurs périphériques (ordinateurs), dans lesquels les périphériques sont tous connectés les uns aux autres à l'aide d'un réseau.

Le package java.net des API Java Standard contient une collection de classes et d'interfaces qui fournissent les détails de communication de bas niveau, permettant d'écrire des programmes qui se concentrent sur la résolution du problème.

1) Java Socket
    (a) définition : Une socket est un point d'extrémité d'une liaison de communication bidirectionnelle entre deux programmes exécutés sur le réseau.Une Socket est utilisée pour la communication entre les applications s'exécutant sur différents JRE. Les classes de socket sont utilisées pour représenter la connexion entre un programme client et un programme serveur. Le paquet java.net fournit deux classes - Socket et ServerSocket - qui implémentent le côté client de la connexion et le côté serveur de la connexion, respectivement.
    (b) La programmation Java Socket peut être orientée connexion ou sans connexion.
    (c) Les classes **Socket** et **ServerSocket** sont utilisées pour la programmation de socket orientée connexion
    (d) et les classes **DatagramSocket** et **DatagramPacket** sont utilisées pour la programmation de socket sans connexion.
    (e) Le client dans la programmation socket doit connaître deux informations : l'adresse IP du serveur et son numéro de port

# Programmation avancée Java - Réseau

| Méthodes de la Classe ServerSocket | | Méthodes de la Classe Socket | |
|---|---|---|---|
| **public InputStream getInputStream()** retourne l'objet InputStream attaché au socket | **public synchronized void close()** ferme la socket | **public Socket accept()** retourne la socket et établit une connexion entre le serveur et le client. | |
| **public OutputStream getOutputStream()** retourne l'objet OutputStream attaché au socket | | **public synchronized void close()** ferme la socket du serveur | |

Exemple de programme serveur utilisant les sockets:

```
import java.io.*;
import java.net.*;
public class MonServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(9999);
Socket s=ss.accept();//établir une connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Exemple de programme client utilisant les sockets:

```
import java.io.*;
import java.net.*;
public class MonClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",9999);
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```


# Questions

Mr. Boubou CAMARA

boubou.camara@yahoo.ca