«640Ko est suffisant pour tout le monde.»
Bill GATES, PDG et fondateur de Microsoft, 1981

1.) Paquetages.

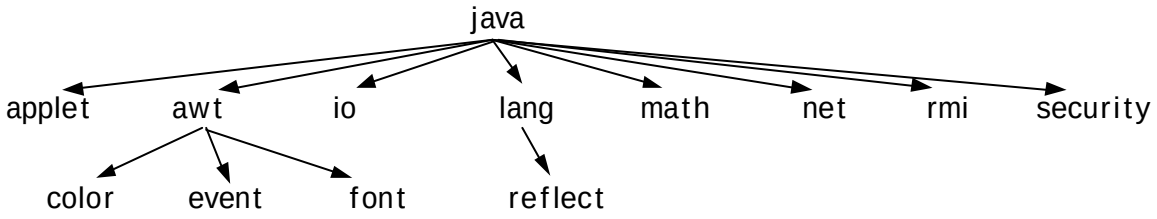
Un paquetage (package en anglais) est un regroupement de classes. On peut aussi le voir comme une bibliothèque de classes. La notion de paquetage introduit un classement hiérarchique des classes qui offre ainsi :

- Un rangement des classes.
- Une nette amélioration de la sécurité d'utilisation des classes en venant renforcer le mécanisme de portée.

On regroupe dans un même paquetage des classes ayant une thématique et des fonctionnalités communes. Un paquetage contient des classes et des sous-paquetages, de la même manière qu'un répertoire contient des fichiers et des sous-répertoires. D'ailleurs, l'unité d'accueil du paquetage est le répertoire et le nom du paquetage est en relation directe avec le nom du paquetage qu'il accueille.

Voici quelques-uns des principaux paquetages fournis avec Java :

java.applet	Classes de base pour les applets
java.awt	Classes d'interface graphique AWT
java.awt.color	Classes pour les couleurs.
java.awt.event	Classes et interfaces de gestion des événements.
java.awt.font	Classes pour les polices de caractères.
java.io	Classes d'entrées/sorties (flux, fichiers…)
java.lang	Classes de support du langage
java.lang.reflect	Classes pour l'examen des classes.
java.math	Classes permettant la gestion de grands nombres.
java.net	Classes de support réseau (URL, *sockets*…)
java.rmi	Classes pour les méthodes invoquées à partir de machines virtuelles non locales.
java.security	Classes et interfaces pour la gestion de la sécurité.
java.sql	Classes pour l'utilisation de JDBC.
java.text	Classes pour la manipulation de textes, de dates et de nombres dans plusieurs langages.
java.util	Classes d'utilitaires (vecteurs, *hashtable* …)
javax.swing	Classes d'interface graphique SWING



Le paquetage java.lang est le plus important (et il est importé par défaut dans un programme) puisqu'il comprend notamment :

- la classe **Object** (superclasse, ancêtre de toutes les classes) fournit entre autres des *méthodes* permettant de copier des objets, de tester l'égalité et de convertir les objets en chaînes;
- les *wrappers* de types simples. A l'instar du C++ et à l'opposé du SmallTalk, pour des soucis de performances, les types simples existent, en dehors des hiérarchies objets. Ces *wrappers* servent de version en classes des types fondamentaux. Il trouvent notamment leurs utilité avec les classes de `java.util` qui prennent comme paramètres des objets;
- la classe **Math** qui permet l'accès aux constantes mathématiques (*pi*, *e*) et aux méthodes statiques (équivalentes aux fonctions du C) implementants les classiques fonctions mathématiques (telles **sqrt**, **sin**…);
- la classe **String**;
- les classes **System** et **Runtime** sont des classes *finales* (qui sont constituées de méthodes statiques a.k.a. fonctions) et permettent d'accéder aux ressources systèmes (notamment les *stdout* et *stdin*) ainsi qu'a l'environnement *runtime*;
- les classes et interfaces de gestions de *threads*, dont la classe **Thread** et l'interface **Runnable**;
- les classes **Class** et **ClassLoader** pour travailler avec les classes. C'est la base du dynamisme Java;
- les classes et interfaces de gestion d'erreurs et d'exceptions;
- les classes de gestion de processus.

Pour utiliser une classe prédéfinie dans un programme Java, le compilateur doit savoir où la trouver. Bien sûr, si la classe est définie dans le même fichier que le programme, il suffit simplement de la référencer. Par contre, si la classe est définie dans un autre fichier (c'est le cas des classes prédéfinies du langage) il est nécessaire d'indiquer au compilateur, de manière non ambiguë, où la trouver. Pour cela, on utilise la clause **import**.

Cette clause **import** demande au compilateur d'inclure un package (c'est à dire une bibliothèque de classes) dans le programme.
Pour qu'une classe appartienne à un paquetage, il faut déclarer ce paquetage au début du fichier source (dans la première ligne utile). Par exemple :

```
package mes_classes;
```

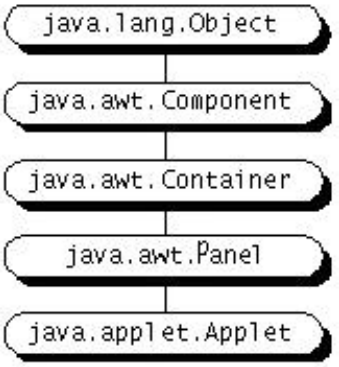
Par convention, les noms de paquetage sont toujours en lettres minuscules.

Si aucun paquetage n'est déclaré, la classe appartient quand même à un paquetage appelé paquetage par défaut. Il est constitué de l'ensemble des classes appartenant au répertoire courant.

Les paquetages sont parfois rassemblés et compactés dans un fichier d'archive au format ZIP ou JAR.

2.) Les applets.

Une **applet** est un programme Java compilé qui est exécuté par le logiciel AppletViewer ou une navigateur[1] WWW qui supporte Java. Les applets n'ont pas été conçues pour l'Internet mais pour les PDA, ces petits ordinateurs portables tels Newton ou General Magic. Mais les caractéristiques de base des applets en font un outil idéal pour l'Internet. En effets celles-ci sont petites, peuvent être télécharger et exécuter très rapidement. Elles ajoutent un composant interactif et fonctionnel aux pages WWW. Par contre elles ont un accès limité aux ressources réseaux et aux systèmes de fichiers locaux pour des raisons de sécurité (Elles deviendraient dans le cas contraire un vecteurs idéal pour les virus). Les navigateurs servent pratiquement de système d'exploitation aux applets. C'est pour cette raison que le succès de Java est tellement lié au succès de ces navigateurs.



Toutes les applets sont implémentés par héritage de la classe Applet. La figure ci-dessus indique la l'arbre d'héritage de la classe Applet. Cette hiérarchie détermine ce qu'une applet peut faire et comment elle peut le faire.

2.1.) Une applet élémentaire.

Ci-dessous, se trouve le code source pour une applet élémentaire appelée Simple. Cette applet affiche une chaîne décrivant à quelle étape se trouve l'applet.

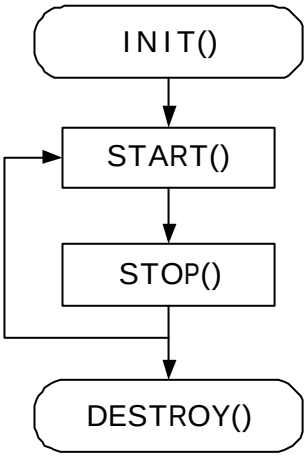
[1] Appelé *butineur* pour les canadiens et *browser* pour les anglophones.

```
import java.applet.Applet;
import java.awt.Graphics;
public class Simple extends Applet {
    StringBuffer buffer;
    public void init() {
        buffer = new StringBuffer();
        addItem("initializing... ");
    }
    public void start() {
        addItem("starting... ");
    }
    public void stop() {
        addItem("stopping... ");
    }
    public void destroy() {
        addItem("preparing for unloading...");
    }
    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }
    public void paint(Graphics g) {
        g.drawRect(0, 0, size().width - 1, size().height -
1);
        g.drawString(buffer.toString(), 5, 15);
    }
}
```

2.2.) Le cycle de vie d'une applet.

Le cycle de vie d'une applet contient quatre phases :

1. L'initialisation (init) quand l'applet est chargée dans le navigateur.
2. Le démarrage (start) quand l'applet est affichée.
3. L'arrêt (stop) quand l'applet cesse d'être affichée.
4. La destruction (destroy) quand l'applet est déchargée du navigateur.



La classe Applet, propose les méthodes suivantes pour gérer le cycle de vie de l'Applet.

Méthode	Description
init()	Cette méthode est appelée lors de la création de l'applet.
start()	Cette méthode est appelée à chaque fois que l'applet est affichée.
stop()	Cette méthode est appelée quand l'utilisateur quitte la page. Quand l'utilisateur quitte la page, l'applet peut s'arrêter et reprendre quand l'utilisateur revient sur la page.
destroy()	Cette méthode est appelée quand l'applet est déchargée de la page.

2.3.) Des méthodes pour dessiner.

Les applets héritent des méthodes pour dessiner proposées par AWT. Le dessin se fait en modifiant la méthode paint comme dans l'applet ci contre :

```
import java.applet.*;
import java.awt.*;
public class Simple extends Applet {
public void fleur(int k, double h, double cx, double cy,
double r, double ad, Graphics g)
{
 int x = 0;
 int y = 0;
 int ox = 0, oy = 0;
 for (int i = 0 ;i<k;i++) {
 x =(int)(cx + (r*Math.cos(2*i*h*Math.PI/k+ad)));
 y =(int)(cy + (r*Math.sin(2*i*h*Math.PI/k+ad)));
 if (i==0) {ox = x; oy = y;} else g.drawLine(ox, oy, x, y);
 }
}
public void paint(Graphics g) {
 g.setColor(Color.pink);
 double np = 200;
 double cx = size().width/2;
 double cy = size().height/2;
 double r = np * 0.27;
 double ad = Math.PI / 2;
 double x = 0;
 double y = 0;
 int k = 15;
 for (int i = 0;i<k;i++) {
 x = cx + r * Math.cos(2*i*Math.PI/k+ad);
 y = cy + r * Math.sin(2*i*Math.PI/k+ad);
 fleur (25,12,x,y,(np*0.22),Math.PI/2,g);
    }
 Font f = new Font("SansSerif",Font.BOLD, 24);
 g.setFont(f);
 g.setColor(Color.blue);
 g.drawString("Je meurs de soif auprès de la
fontaine...",10,size().height/2);
}
}
```

Pour dessiner, on utilise donc un objet de la classe Graphics qui est passé en paramètre à la méthode paint. Voici quelques-unes unes des méthodes de la classe Graphics.

Methodes	
`abstract void`	clearRect(`int x, int y, int width, int height`) Efface le rectangle spécifié en le remplissant avec la couleur de fond.
`abstract void`	copyArea(`int x, int y, int width, int height, int dx, int dy`) Copie une zone à une distance dx dx.
`void`	draw3DRect(`int x, int y, int width, int height, boolean raised`) Dessine un regtangle en 3D.
`abstract void`	drawArc(`int x, int y, int width, int height, int startAngle, intarcAngle`) Dessine un arc.

`abstract boolean`	drawImage(Image `img, int x, int y,` Color `bgcolor,` ImageObserver `observer)` Draws as much of the specified image as is currently available.
`abstract void`	drawLine(`int x1, int y1, int x2, int y2)` Dessine une ligne entre les points (x1,y1) et (x2,y2).
`abstract void`	drawOval(`int x, int y, int width, int height)` Dessine un ovale.
`abstract void`	drawPolygon(`int[] xPoints, int[] yPoints, int nPoints)` Dessine un polygone à partir de tableaux de points.
`void`	drawPolygon(Polygon `p)` Dessine un polygone à partir d'un objet Polygon. .
`void`	drawRect(`int x, int y, int width, int height)` Dessine un rectangle.
`abstract void`	drawRoundRect(`int x, int y, int width, int height, int arcWidth, int arcHeight)` Dessine le contour d'un rectangle aux coins arrondis.
`abstract void`	drawString(String `str, int x, int y)` Dessine une chaîne.
`void`	fill3DRect(`int x, int y, int width, int height, boolean raised)` Rempli un rectangle en 3D.
`abstract void`	fillArc(`int x, int y, int width, int height, int startAngle, int arcAngle)` Rempli un arc..
`abstract void`	fillOval(`int x, int y, int width, int height)` Rempli.un ovale.
`abstract void`	fillPolygon(`int[] xPoints, int[] yPoints, int nPoints)` Rempli un polygone fermé.
`void`	fillPolygon(Polygon `p)` Rempli un polygone fermé.
`abstract void`	fillRect(`int x, int y, int width, int height)` Rempli un rectangle.
`abstract void`	fillRoundRect(`int x, int y, int width, int height, int arcWidth, int arcHeight)` Rempli un rectangle arrondi.
`abstract Color`	getColor`()` Remvoi la couleur courante.
`abstract Font`	getFont`()` Renvoie la police courante.
`abstract void`	setColor(Color `c)` Fixe la couleur courante.
`abstract void`	setFont(Font `font)` Fixe la police courante.
`abstract void`	setXORMode(Color `c1)` Fixe le mode graphique en XOR.

2.4.) Intégrer une Applet dans une page HTML.

Pour faire fonctionner une applet, il faut la placer dans une page Web et visualiser cette page avec un navigateur supportant Java. Pour placer une applet dans une page Web, on utilise une balise

spéciale pour indiquer à la page qu'elle doit charger et lancer une applet. La balise de base se présente ainsi :

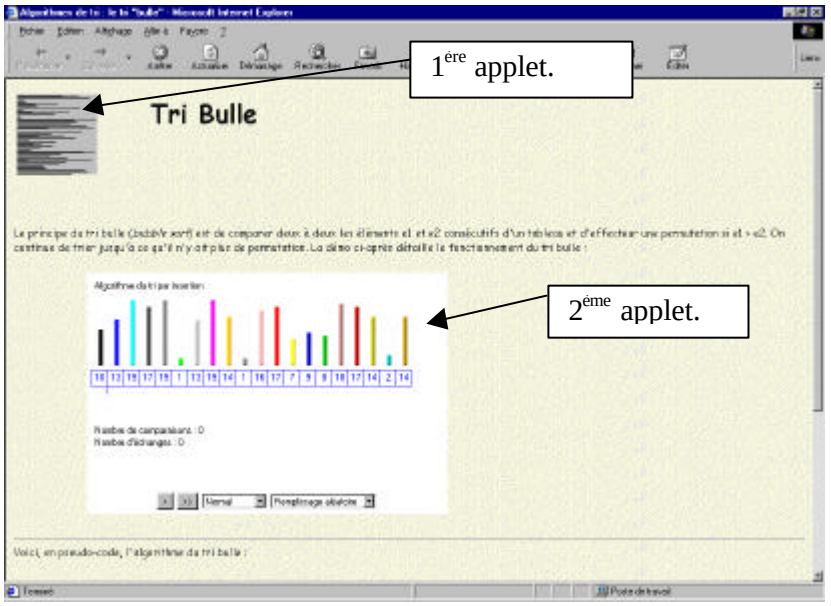
```
<html>

<applet codebase=EndroitOuTrouverLeCode code=nomDuFichier.class
width=largeur height=hauteur alt=CodeHTMLdeRemplacement>
<param name="Paramètre1" value="valeurDuParamètre"
<param name="Paramètre2" value="valeurDuParamètre"

<param name="ParamètreX" value="valeurDuParamètre"
</applet>

</html>
```

L'applet sera donc exécutée par le navigateur Web (ou l'AppletViewer) à l'endroit ou se trouve le *tag* (*inline*) et avec les paramètres physiques (*width*, *height* et autres) spécifiés et logiques (param1…x). Voici un exemple de page html contenant 2 applets :



On peut aussi visualiser une applet en utilisant l'utilitaire « Appletviewer » fourni avec le JDK. Une manière simple de l'utiliser et d'insérer dans le code source du programme le commentaire suivant :

```
// Embedding the applet tag for Appletviewer
// <applet code=MyApplet width=200 height=100>
// </applet>
```

puis de lancer le programme Appletviewer ainsi :

```
Appletviewer MyApplet.java
```

2.5.) Stocker les fichiers de l'applet dans un fichier « .jar ».

```
<applet code="Tetris.class" align="middle" width=400 height=400
archive = "Tetris.jar">
<param name="images" value="maud.jpg">Tetris</applet>
```

2.6.) Ce qu'une applet peut et ne peut pas faire.

Les applets sont conçues pour être téléchargées depuis un site distant et exécutées localement. Les préoccupations de sécurité sont donc fondamentales. De ce fait, les applets, contrairement aux applications, ont des possibilités d'exécutions restreintes.

Droits des programmes Java	**Applet dans un navigateur**	**Visualisateur d'applets**	**Java exécutant une application**
Lire les fichiers locaux	Non	Oui	Oui
Ecrire dans des fichiers locaux	Non	Oui	Oui
Obtenir des informations sur des fichiers	Non	Oui	Oui
Supprimer un fichier	Non	Non	Oui
Lancer un autre programme	Non	Oui	Oui
Se connecter au port réseau du serveur	Oui	Oui	Oui
Se connecter au port réseau d'un autre hôte.	Non	Oui	Oui
Charger la bibliothèque Java	Non	Oui	Oui
Sortir de l'applet par un exit	Non	Oui	Oui
Créer une fenêtre popup.	Oui avec avertissement	Oui	Oui

2.7.) La classe java.applet.Applet.

Constructeurs	
Applet( )	

Methodes	
`void`	destroy( ) Cette méthode est appelée par le navigateur ou par le visualisateur d'applet pour l'informer que celle-ci va être détruite.
`AppletContext`	getAppletContext( ) Renvoie le contexte d'exécution de l'applet. La connaissance de ce contexte va permettre à l'applet d'agir sur son environnement (changer de page html par exemple).
`String`	getAppletInfo( ) Renvoie des infos sur cette applet.
`AudioClip`	getAudioClip(URL `url`) Renvoie le clip audio spécifié par url.
`AudioClip`	getAudioClip(URL `url`, String `name`) Renvoie le clip audio spécifié par url et name.
`URL`	getCodeBase( ) Renvoie l'URL à partir de laquelle est chargée l'applet.
`URL`	getDocumentBase( ) Renvoie l'URL du document.
`Image`	getImage(URL `url`) Renvoie l'objet image spécifié dans url.
`Image`	getImage(URL `url`, String `name`) Renvoie l'objet image spécifié par url et name.
`Locale`	getLocale ( )

`String`	getParameter(String `name`) Renvoie le paramètre de nom name spécifié dans la balise html.
`String[][]`	getParameterInfo( ) Renvoie des informations sur les paramètres.
`void`	init( ) Appelée par le navigateur pour informer l'applet qu'elle est chargée par le système.
`boolean`	isActive( ) Renvoie vrai si l'applet est active.
`void`	resize(`int width, int height`) Demande à ce que l'applet soit agrandie
`void`	showStatus(String `msg`) Demande que msg soit affiché dans la barre de commentaire du navigateur.
`void`	start( ) Appelée par le navigateur pour informer l'applet qu'elle devient active.
`void`	stop( ) Appelée par le navigateur pour informer l'applet qu'elle devient inactive.

2.8.) L'applet GraphApplet.

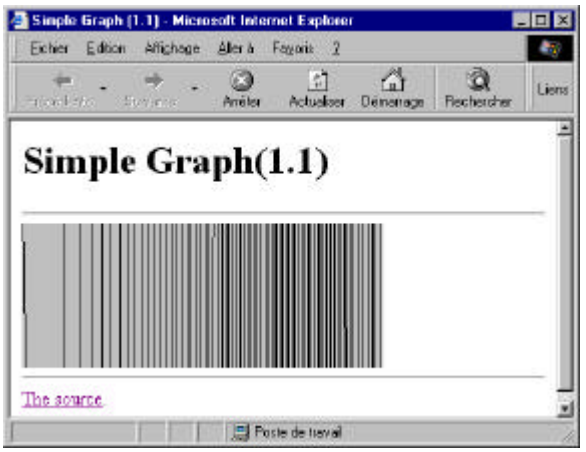
L'applet GraphApplet.java est fournie avec les exemples du jdk1.2. Elle se trouve dans le répertoire `/demo/applet/SimpleGraph`. En voici le code source :

```
import java.awt.Graphics;
public class GraphApplet extends java.applet.Applet {
  double f(double x) {
    return (x*x) * getSize().height / 4;
  }
  public void paint(Graphics g) {
    for (int x = 0 ; x < getSize().width ; x++) {
      g.drawLine(x, (int)f(x), x + 1, (int)f(x + 1));
    }
  }
  public String getAppletInfo(){
    return "Draws a sin graph.";
  }
}
```

Elle est utilisée dans le fichier example1.html. En voici le code :

```
<html>
  <head>
      <title>Simple Graph (1.1)</title>
  </head>
  <body>
      <h1>Simple Graph(1.1)</h1>
      <hr>
      <applet code=GraphApplet.class width=300 height=120>
      alt="Your  browser  understands  the  &lt;APPLET&gt;  tag
but
      isn't running the applet, for some reason."
      Your browser is completely ignoring the &lt;APPLET&gt;
      tag!
      </applet>
      <hr>
      <a href="GraphApplet.java">The source</a>.
  </body>
</html>
```

Et voici le résultat :



2.9.) Le « double buffering »

Pour se dessiner à l'écran, les applets utilisent deux métodes : Update (Graphics g) et paint (Graphics g).

Pour éviter les effets de scintillement, on utilise un buffer :

```
Image buffer ;

public void init() {
      buffer = createImage(100,100);
}
public void paint(Graphics g) {
      update(g);
}
public void update(Graphics g) {
      Graphics g1  = buffer.getGraphics() ;
      g1.drawOval(10,10,10,10) ;
      g.drawImage(buffer,0,0,700,430,null);
}
```

2.10.) Chargement d'une ressource.

2.10.1.) CHARGEMENT D'UNE IMAGE.

```
MediaTracker tracker = new MediaTracker(this);
dessin = getImage(getCodeBase(),"mon_image.gif");
if (dessin != null) {
      tracker.addImage(dessin, 0);
      try {
            tracker.waitForID(0);
      } catch (InterruptedException ex) {dessin = null;}
}
```

2.10.2.) CHARGEMENT D'UN FICHIER.

```
public void chargement() {

      BufferedReader fichier;
      try {
            InputStream in =
            Getclass().getResourceAsStream("nom_de_fichier.txt");
            fichier = new BufferedReader(new InputStreamReader(in));
            s = new String();
            while((s = fichier.readLine())!= null) {
                  s = s.trim();
                  s = s.toUpperCase();
            }
            fichier.close();
            fichier = null;
      }
            catch (Exception e) {
            fichier = null;
      }
```

2.11.) Transformer une applet en application.

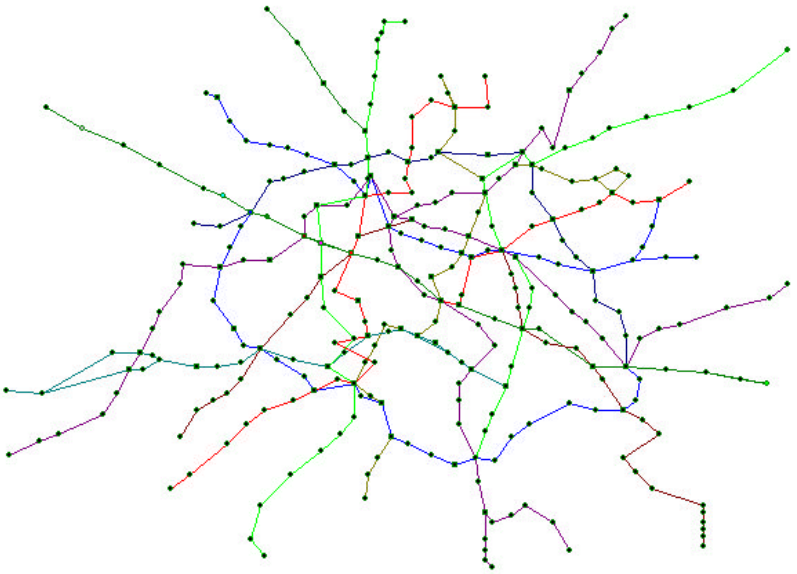
3.) Les graphes.

Les graphes représentent un instrument puissant pour modéliser de nombreux problèmes combinatoires, qui seraient sans cela difficilement abordables par des techniques classiques comme l'analyse mathématique. En plus de son existence en tant qu'objet mathématique, le graphe est aussi une structure de données puissante pour l'informatique.

Les graphes sont irremplaçables dès qu'il s'agit de décrire la structure d'un ensemble complexe, en exprimant les relations, les dépendances entre ses éléments. Des exemples sont les diagrammes hiérarchiques en sociologie, les arbres généalogiques, les arbres phylogénétiques et les diagrammes de succession de tâches en gestion de projets.

Un autre grand groupe d'utilisation est la représentation de connexions et de possibilités de cheminement : détermination de la connexité d'un réseau quelconque (électrique, d'adduction d'eau...), calcul du plus court chemin dans un réseau routier etc.

Les graphes sont enfin des outils précieux pour décrire des systèmes dynamiques, c'est à dire évoluant dans le temps : diagrammes de transition, automates d'états finis, chaînes de Markov.



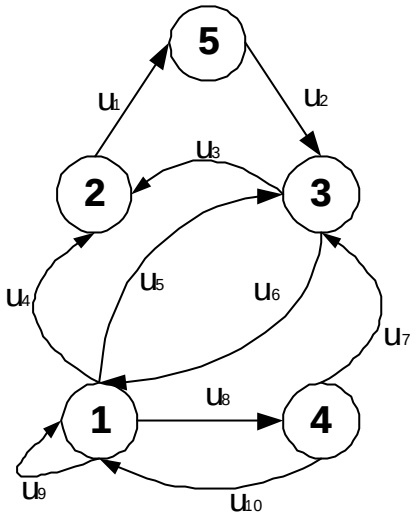
Graphe du métro parisien.

3.1.) Graphes orientés.

Un **graphe orienté** G est défini par un doublet [X,U] où :

- X est l'ensemble des sommets (ou noeuds) du graphe.
- U est l'ensemble des arcs.

Exemple :



G = [X,U] où :

X = {1,2,3,4,5}

U = {U_1,U_2,U_3,U_4,U_5,U_6,U_7,U_8,U_9,U_{10}}

- Un **chemin** d'un sommet X_1 vers un sommet X_n est une suite $X_1,X_2,X_3,X_4...X_n$ telle que pour tout i<n il existe un arc de X_i vers X_n.

- X_1 est l'**extrémité initiale** du chemin, X_n est l'**extrémité terminale** du chemin. Exemple de chemin reliant le sommet 2 au sommet 4 : P = {2,5,3,1,4}.

- La **longueur du chemin** est égale au nombre de sommets - 1 composants le chemin. La longueur du chemin P = {2,5,3,1,4} est donc égale à 4.

- Un chemin de longueur q (de cardinalité q) est une séquence de q arcs : P = {u_1,u_2,u_6,u_8}

- Un **chemin élémentaire** est un chemin tel qu'en le parcourant, on ne rencontre pas deux fois le même sommet.

- Un **circuit** est un chemin d'un sommet vers lui-même. Par exemple, P = {1,2,5,3,1} est un circuit.

- Un **circuit élémentaire** est un circuit tel qu'en le parcourant, on ne rencontre pas deux fois le même sommet (sauf le sommet d'origine).

- Une **boucle** est un circuit de longueur 1. Par exemple, P={1,1} est une boucle.

- Un **chemin simple** est un chemin tel qu'en le parcourant, on ne passe pas deux fois par le même arc.
- Si dans un graphe G=[X,U] il existe un arc du sommet X_i vers le sommet X_j, alors X_i est le prédécesseur de X_j et X_j est le successeur de X_i.

- Un **P-Graphe** est un graphe dans lequel il n'existe jamais plus de P arcs entre deux sommets X_i et X_j. Un 1-Graphe est un graphe dans lequel il n'existe jamais plus d'un arc entre deux sommets.

- Un graphe est **complet** si toute paire de sommets est connectée par un arc (ou une arête).

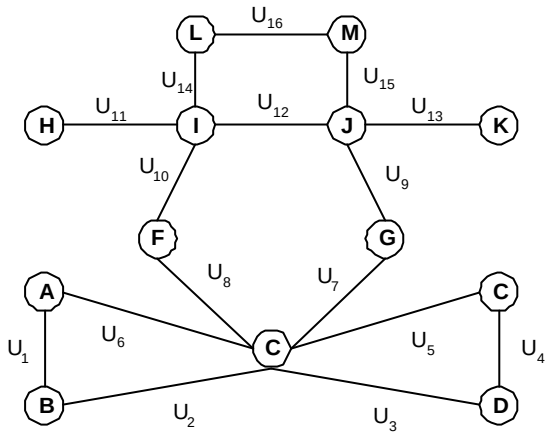
- Un graphe est **symétrique** si l'existence d'un arc du sommet X_i vers le sommet X_j implique l'existence d'un arc du sommet X_j vers le sommet X_i.

3.2.) Graphes non orientés.

Un graphe non orienté G est défini par un doublet [X,U] où :

- X représente l'ensemble des sommets des arcs.
- U représente l'ensemble des arrêtes du graphe.

Exemple :



G = [X,U] où

- X = {A ,B,C,D,E,F,G,H,I,J,K,L,M}
- U = {U_1,U_2,U_3,U_4,U_5,U_6,U_7, U_8,U_9,U_{10},U_{11},U_{12},U_{13},U_{14}, U_{15},U_{16}}

- Une chaîne est une séquence d'arrêtes consécutives. Exemple : P = {U_1,U_2,U_5}.

- Un cycle est une chaîne fermée (une chaîne d'un sommet vers lui-même). Exemple : P = {U_4,U_1,U_2,U_5,U_6}.

- On appelle chaîne élémentaire une chaîne telle qu'en la parcourant, on ne rencontre pas deux fois le même sommet.

- On appelle cycle élémentaire un cycle tel qu'en le parcourant, on ne rencontre pas deux fois le même sommet, sauf le sommet d'origine.

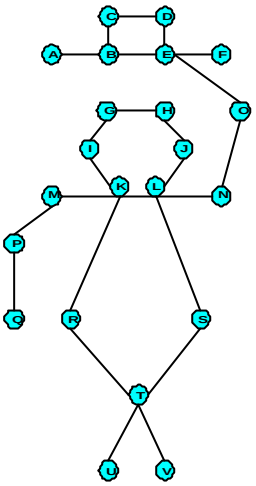
- Un multigraphe est un graphe pour lequel il peut exister plusieurs arrêtes entre deux sommets X_i et X_j.

- Un graphe est dit simple si :

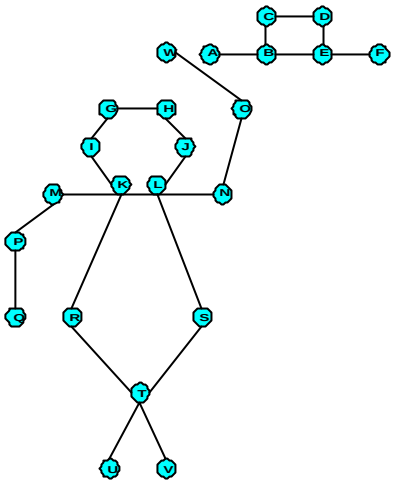
  - Il est sans boucle.
  - Il n'y a pas plus d'une arrête entre deux sommets quelconques.

3.3.) Connexité - Forte connexité.

- Un graphe **non orienté** est dit **connexe** si pour tout couple de sommets X_i et X_j, il existe une chaîne joignant X_i et X_j.



Un graphe connexe



Un graphe non connexe

- Un graphe **orienté** est dit **fortement connexe** s'il existe un chemin entre toute paire de sommets.

3.4.) Concepts importants.

- Un **circuit hamiltonien** (cycle hamiltonien) est un circuit (cycle) passant une fois et une seule fois par chacun des sommets d'un graphe. Un graphe est dit hamiltonien s'il contient un circuit (cycle) hamiltonien.

- Une chaîne **eulérienne** est une chaîne empruntant une fois et une seule fois chaque arrête d'un graphe. Un cycle eulérien est une chaîne eulérienne dont les extrémités coïncident.

- Le **degré d'un sommet** est le nombre d'arcs distincts ayant une et une seule extrémité en ce sommet.

- Le **demi-degré extérieur** d'un sommet X_i (noté $d^+(Xi)$) est le nombre d'arcs ayant comme extrémité initiale le sommet X_i et comme extrémité finale un sommet X_j avec $X_i <> X_j$.

- Le **demi-degré intérieur** d'un sommet X_i (noté $d^-(X_i)$) est le nombre d'arcs ayant comme extrémité finale le sommet Xi et comme extrémité initiale le sommet X_j avec $X_i <> X_j$.

- Pour les **graphes non orientés**, le **degré d'un sommet** est le nombre d'arrêtes distinctes ayant une et une seule extrémité en ce sommet.

- Deux arcs (arêtes) sont dits **adjacents** s'ils ont au moins une extrémité commune.

- La densité d'un graphe est le rapport entre le nombre d'arcs (ou d'arêtes) que compte le graphe et le nombre d'arcs (ou d'arêtes) que comporterait un graphe complet ayant le même nombre de sommets. On aura donc :

    - Pour un graphe orienté :

$$densité = \frac{nombre\_d\_arcs}{nombre\_de\_sommets * (nombre\_de\_sommets - 1)}$$

- Pour un graphe non orienté :

$$densité = \frac{2 * nombre\_d\_arêtes}{nombre\_de\_sommets * (nombre\_de\_sommets - 1)}$$

3.5.) Représentation des graphes.

3.5.1.) MATRICE D'ADJACENCE.

Soit le graphe G=[X,U]. On peut associer à ce graphe une **matrice d'adjacence** dont les éléments seront de la forme a(i,j) avec :

- i variant de 1 à N (N = nombre de sommets).
- j variant de 1 à N.
- $(x_i,x_j) \in U \Rightarrow a_{ij} = 1$.
- $(x_i,x_j) \notin U \Rightarrow a_{ij} = 0$.

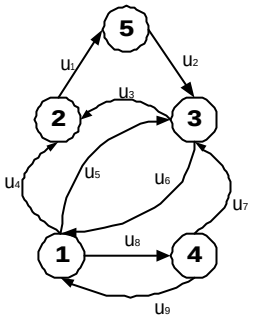
Exemple :

Pour le graphe orienté G=[X,U]

X = {1,2,3,4,5}

U = {U_1,U_2,U_3,U_4,U_5,U_6,U_7,U_8,U_9}

N = 5



On aura la matrice d'adjacence A =

	1	**2**	**3**	**4**	**5**
1	0	1	1	1	0
2	0	0	0	0	1
3	1	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0

3.5.2.) MATRICE D'INCIDENCE.

Soit le graphe G = [X,U]. On peut associer à ce graphe une **matrice d'incidence** dont les éléments sont de la forme a(i,j) avec :

- i variant de 1 à N (N=nombre de sommets).
- j variant de 1 à M (M=nombre d'arcs).

A chaque ligne de la matrice correspond un sommet. A chaque colonne correspond un arc. On aura :

- $a_{iu} = +1$ si $u \in w^+(i)$.
- $a_{iu} = -1$ si $u \in w^-(i)$.
- $a_{iu} = 0$ si $u \notin w(i)$.

Par exemple, si on reprend le graphe précédent on aura :

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
1	0	0	0	-1	-1	+1	0	-1	+1
2	-1	0	0	+1	0	0	0	0	0
3	0	+1	-1	0	+1	-1	+1	0	0
4	0	0	0	0	0	0	-1	+1	-1
5	+1	-1	+1	0	0	0	0	0	0

Ce type de représentation n'est adapté qu'aux graphes ne contenant pas de boucle.

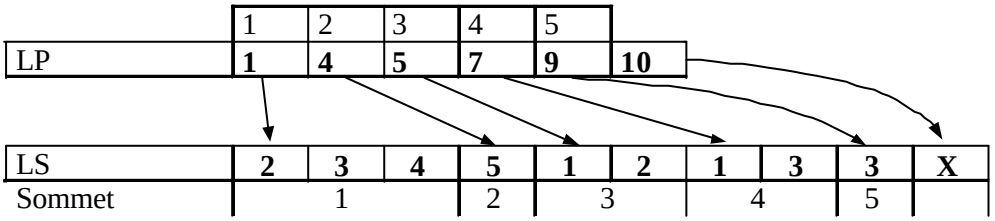
3.5.3.) LISTE D'ADJACENCE.

On adopte la représentation par **liste d'adjacence** dans le cas de graphes peut denses, pour éviter une trop grande perte de place avec les représentations en matrice.

On utilisera deux listes :

- LP : Liste des pointeurs de dimension N+1 (n=nombre de sommets).
- LS : Liste des sommets de dimension M+1 (M=nombre d'arcs).

Par exemple, on représentera le graphe G=[X,U] de la manière suivante :



Les informations relatives au sommet n sont donc comprises entre LS(LP(n)) et LS(LP(n+1)-1).

3.6.) Modélisation « Orientée Objet ».

4.) Exercices.

Exercice 5.2 :

A LAVAL IL AVALA est un palindrome, tout comme 239 932 ou UBU. Un palindrome est en effet une phrase, un nombre ou un message pouvant être lu indifféremment de gauche à droite ou de droite à gauche.

Je rappelle qu'un nombre entier palindrome ayant un nombre pair de chiffres est toujours divisible par 11, mais que la réciproque est fausse (par exemple, 2211, évidemment multiple de 11, qui possède 4 chiffres, n'est pas un palindrome).

Ainsi, 239 932 = 11 * 21812

On peut vérifier aussi que 21812, bien que palindrome, n' est pas divisible par 11 (il aurait pu l'être, mais ce n'était pas obligatoire car le nombre de ses chiffres, 5, est impair)

Déterminer tous les nombres premiers inférieurs à 10 000 qui sont des palindromes et dont la somme des chiffres est un multiple de 11.

Exercice 5.2 :

Léon Noël, un habitant de Laval, est obsédé par les palindromes ; sa dernière trouvaille: un nombre palindrome de trois chiffres qui est la somme de deux nombres palindromes de deux chiffres. Quel est ce nombre palindrome de 3 chiffres ?