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

1.) Les objets dans Java.

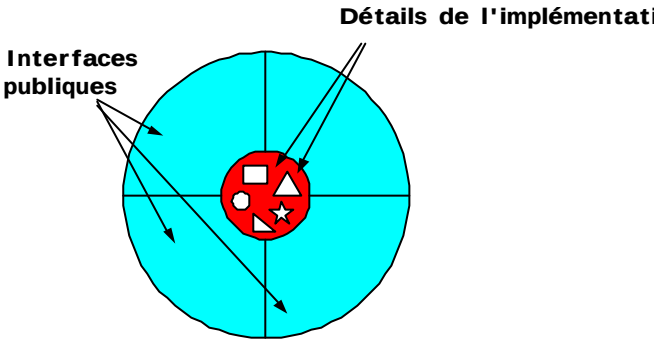
1.1.) Rappels.

1.1.1.) QU'EST CE QU'UN OBJET ?

Comme le nom de "Programmation Orientée Objet" l'indique clairement, les objets sont au centre des technologies orientées objet. Les objets utilisés en POO possèdent des attributs (encore appelés champs) et des méthodes. On peut donc définir un objet de la manière suivante :

Un objet est une association logicielle de données et de méthodes.

L'illustration suivante est une représentation habituelle d'un objet logiciel :



Tout ce que le composant logiciel connaît (les données) et peut faire (les méthodes) est exprimé aux travers des variables et des méthodes. Inversement, ce que l'objet ne peut pas connaître et ne peut pas faire doit être exclus de l'objet. Dans beaucoup de langages - y compris Java - le concepteur d'un objet peut malgré tout choisir de permettre l'accès à ses données à d'autres objets. De la même manière, il peut interdire l'accès à certaines méthodes.

L'idée d'encapsuler les données et les méthodes apporte les avantages suivants :

- Plus de modularité. Le code source d'un objet peut être maintenu indépendamment du code source des autres objets tant que son interface n'est pas touchée.
- Le masquage des informations dont l'utilisateur de l'objet n'a pas besoin et/ou qu'il ne doit pas pouvoir manipuler.

1.1.2.) QU'EST CE QU'UN MESSAGE ?

Un programme est donc constitué d'un assemblage d'objets. Ces objets doivent pouvoir communiquer entre eux. Les interactions entre objets s'appellent des messages.

- Le destinataire du message.
- Le nom de la méthode à exécuter.
- Les paramètres demandés par la méthode.

Les avantages de ce mode d'interaction par messages sont :

- La manipulation d'objets se fait uniquement par l'appel de méthodes et non par accès direct aux données.
- Les objets n'ont pas besoin de faire partie du même processus (ni même s'exécuter sur la même machine) pour interagir entre eux.

1.1.3.) QU'EST CE QU'UNE CLASSE ?

Une classe c'est un modèle (ou un prototype) qui défini les variables et les méthodes communes à tous les objets d'un certain type.

Les valeurs de chaque donnée peuvent être différentes pour chaque instance d'une classe (donc pour chaque objet). La classe décrit les facteurs communs à tous les objets qui peuvent être créé sur son modèle.

Si les objets apportent la modularité des programmes, les classes apportent la réutilisabilité et les mise en facteur des composants logiciels.

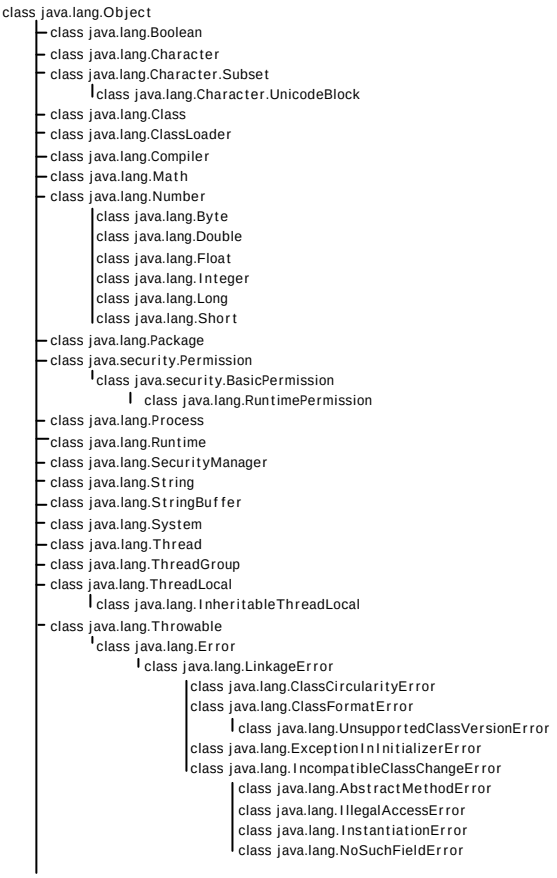
1.1.4.) QU'EST CE QUE L'HERITAGE ?

Les objets sont définis par leurs classes. En connaissant la classe d'un objet, on connaît les services que propose l'objet.

En Programmation Orientée Objet, il est possible de définir une classe en fonction d'une autre classe. Par exemples, la classe des nombres entiers et la classe des nombres réels dérivent toutes deux de la classe des nombres. Dans la terminologie POO, les classes "nombres entiers" et "nombres réels" sont deux sous-classes de la classe des nombres. Inversement, la classe "nombre" est la classe ancêtre (parfois aussi appelée "super-classe") des classes "nombres entier" et "nombres réels".

Chaque sous-classe hérite des propriétés et des méthodes de sa classe ancêtre. Cependant, une sous-classe peut étendre sa classe ancêtre en lui ajoutant des propriétés ou des méthodes ou alors en masquant les propriétés ou les méthodes de la classe ancêtre.

Bien entendu, on n'est pas limité à un seul niveau d'héritage. L'arbre d'héritage peut être aussi profond que nécessaire. Le schéma ci-contre donne une partie de l'arbre d'héritage des classes du package java.lang.



1.2.) Le cycle de vie d'un objet.

Un programme Java est donc constitué d'un ensemble d'objets créés sur le modèle de différentes classes. Ces objets interagissent entre eux par envoi de messages.

1.2.1.) CREATION D'OBJETS.

En Java, on créé des objets en créant une instance d'une classe, c'est à dire en instanciant une classe. Le code suivant créé une instance d'un objet BigInteger :

```
BigInteger un_grand_nombre = new BigInteger("123456789101112");
```

Cette simple instruction effectue trois actions :

1. **La déclaration**. L'instruction BigInteger un_grand_nombre est une déclaration de variable qui signale aux compilateur que l'identifiant un_grand_nombre fait référence à un objet de la classe BigInteger.
2. **L'instanciation**. L'opérateur new est l'opérateur qui permet de créé un objet, c'est à dire de lui allouer un espace mémoire.
3. **L'initialisation**. L'instruction BigInteger("123456789101112") est un appel au constructeur de la classe BigInteger qui initialise l'objet.

Bien entendu, comme pour les variables de type primitif, la déclaration peut se faire à part. Par contre, l'instanciation et l'initialisation se font obligatoirement dans la même instruction. On pourrait donc avoir :

```
BigInteger un_grand_nombre;
...
un_grand_nombre = new BigInteger("123456789101112");
```

Une classe peut proposer plusieurs constructeurs différents. Si ceux-ci ont acceptent des arguments différents, ils ont obligatoirement de nom de la classe.

1.2.2.) UTILISATION D'OBJETS.

Une fois l'objet créé, on va l'utiliser. On peut lui demander des informations, le faire changer d'état ou lui faire accomplir des actions. Pour cela, deux possibilités existent :

1. Manipuler ou consulter ses variables. C'est la plus mauvaise méthode et il faut l'utiliser le moins souvent possible. En effet, en manipulant directement ses données, on peut mettre l'objet dans un état incohérent et perturber son fonctionnement.
2. Appeler ses méthodes. C'est la solution à préférer. Le concepteur de l'objet doit s'assurer que ses méthodes laissent toujours l'objet dans un état cohérent et que toutes les exceptions qui pourraient se produire sont bien traitées.

Voici un exemple de manipulation de l'objet "un_grand_nombre" :

```
boolen reponse;
reponse = un_grand_nombre.isProbablePrime(200);
```

Cette instruction demande à l'objet "un_grand_nombre" s'il est probablement un nombre premier. En général, on appellera les méthodes d'un objet par une instruction de la forme :

```
nom_de_l_objet.nom_de_méthode(parametres_de_méthodes);
```

Appeler une méthode d'un objet est équivalent à envoyer un message à un objet.

1.2.3.) SUPPRESSION D'OBJETS.

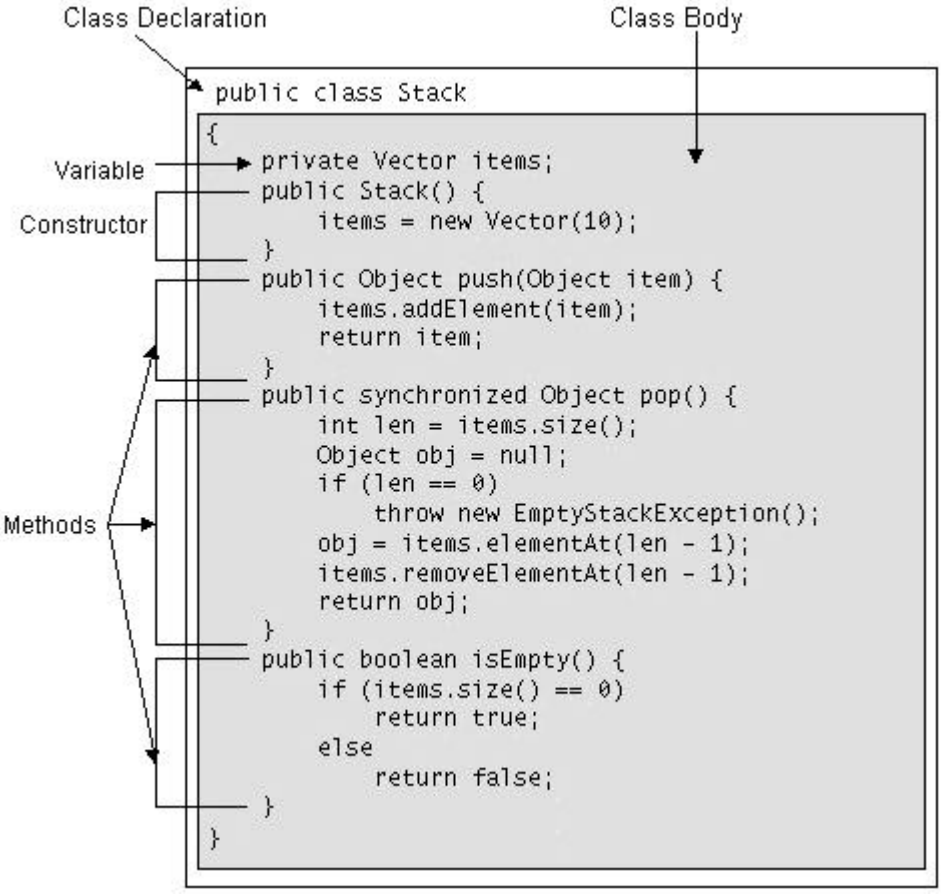
La plupart des langages orientés objets demandent au programmeur de supprimer les objets qu'il a créé une fois qu'il n'en a plus l'usage. Cette technique de management de la mémoire est source d'erreurs fréquentes (oubli de détruire des objets ou utilisation d'objets préalablement détruits). C'est pourquoi le langage Java a été conçu de manière à ce que la destruction soit inutile. Dés qu'un objet n'est plus référencé dans le programme, il est détruit par un processus qu'on appelle le "garbage collector" ou le "ramasse-miettes".

Ce "garbage collector" scanne périodiquement la mémoire utilisée par le programme et détruit les objets devenus inutiles. Pour cela, il parcourt la mémoire et "marque" les objets qui sont référencés par d'autres objets. Ensuite, il supprime ceux qui ne sont pas marqués. Ce processus tourne avec une faible priorité

Avant de détruire un objet, le "garbage collector" appelle toujours sa méthode "finalize()". Cependant, comme on ne sait jamais exactement à quel moment cette méthode sera appelée, on ne la surcharge généralement pas.

1.3.) Les classes.

Le schéma suivant indique la structure d'une classe :



La déclaration de classe comprend donc deux parties : la déclaration proprement type et le corps de la

classe.

1.3.1.) DECLARATION DE CLASSE.

Le coté gauche du tableau suivant montre les différentes composantes possibles de la déclaration d'une classe, dans l'ordre où ils doivent apparaître lors de la déclaration d'une classe.

Le mot clé class et le nom de la classe sont obligatoires. Toutes les autres indications sont optionnelles. Par défaut, la classe ne sera pas publique, ce ne sera pas une classe abstraite ni une classe finale, elle héritera de la classe Object et n'implémentera pas d'interfaces.

public	La classe est accessible à tous
abstract	La classe ne peut être instanciée
final	La classe ne peut être héritée
class *nom_de_classe*	Nom de la classe
extends *Super*	Classe ancêtre
implements *Interfaces*	Interfaces implémentés par la classe
{ corps de la classe **}**	

1.3.2.) CORPS DE LA CLASSE.

Le corps de la classe contient tout le code nécessaire durant tout le cycle de vie de l'objet. On a donc :

- Des constructeurs pour initialiser l'objet.
- La déclaration de variables qui donnent l'état de l'objet.
- Les méthodes qui implémentent les fonctionnalités de l'objet.
- Eventuellement, une méthode finalize qui donnent le code à exécuter lors de la destruction de l'objet.

Les variables et les méthodes sont appelés *membres* de la classe.

La classe Stack définit une variable destinée à contenir les éléments de la pile, il s'agit de l'objet de type Vector appelé *items*. Elle définit un constructeur (ne recevant aucun argument) et trois méthodes push, pop et Empty.

1.3.3.) CONSTRUCTEURS.

Toutes mes classes java possèdent des constructeurs qui sont utilisés pour initialiser un nouvel objet de cette classe. Le constructeur possède toujours le même nom que la classe. Par exemple, le nom du constructeur de la classe Stack s'appelle Stack.

```
public Stack() {
    items = new Vector(10);
}
```

Le langage Java supporte les surcharges pour les constructeurs. Une même classe peut donc avoir un nombre quelconque de constructeurs de même nom. Cependant, la signature du constructeur (nom+liste des arguments) devra être différente. Pour la classe Stack, on aurait pu définir un second constructeur :

```
public Stack(int initialSize) {
    items = new Vector(initialSize);
}
```

Ces deux constructeurs ont le même nom (Stack) mais on une liste de paramètres différente. Le compilateur fera la différence entre ces constructeurs en se basant sur le nombre de paramètres et leurs types.

Généralement, un constructeur utilise ses arguments pour initialiser l'état du nouvel objet. Pour initialiser un objet, on choisit le constructeur dont les arguments reflètent le mieux la manière dont on souhaite que l'objet soit initialisé. A noter que le constructeur par défaut (celui qui ne prend aucun argument) est généré automatiquement par le système pour les classes ne possèdant pas de constructeur. Ce constructeur produit par le système ne procède à aucune initialisation des variables de classe et donc, si on veut que ces variables soit initialisées, il faut écrire un constructeur.

1.3.4.) VARIABLES DE CLASSE.

La classe Stack utilise la ligne suivante pour définir son unique variable de classe :

```
private Vector items;
```

Ceci déclare une variable de classe et non une variable locale car la définition apparaît dans le corps de la classe mais en dehors de toute méthode ou constructeurs. La variable de classe déclarée a pour nom items, et son type est Vector. Le mot-clé private donne à cette variable le statut de donnée privée ce qui fait qu'elle n'est pas accessible en dehors de la classe Stack.

Chaque variable de classe peut possèder les attributs suivants :

1. Niveau d'accès. Une variable peut être public, private, ou protected.
2. static permet de déclaré une variable de classe plutôt qu'une variable d'instance.
3. final indique que la valeur de cette variable ne peut changer.
4. type le type de données.
5. nom. le nom de la variable.

1.3.5.) METHODES.

Une méthode Java peut s'apparenter à un sous-programme. En Java, les méthodes déterminent les messages que l'objet peut traiter.

Une méthode est toujours composée d'au moins quatre éléments : son nom, ses arguments, le type de donnée retournée et le corps de la méthode.

```
returnType methodName( /* argument list */ ) {
  /* Method body */
}
```

Le type de donnée retournée correspond à la valeur que la méthode renverra à la fin de son exécution. Le nom de la méthode permet de l'identifier. La liste des arguments donne les types et les noms des informations que l'on désire passer à la méthode.

En Java, les méthodes ne peuvent être crées qu'à l'intérieur de classes. Une méthode ne peut être appelée que par un objet et cet objet doit posséder l'autorisation d'appeler cette méthode. Si on n'essaie d'appeler une méthode qui n'existe pas ou à laquelle on n'a pas accès, le compilateur produit un message d'erreur.

On appelle une méthode avec la syntaxe suivante :

```
nom_de_l_objet.nom_de_la_méthode(param_1,param_2,...param_n) ;
```

Par exemple, si on a une méthode appelée la_méthode qui n'attend aucun argument et renvoie un entier et un objet appelé un_objet pour lequel la_méthode peut être appelée, on pourra écrire :

```
int x = un_objet.la_méthode();
```

Bien entendu, le type de donné retourné par la_méthode doit être compatible avec x. Cette action d'appel de méthode est appelée, en POO, envoi d'un message à un objet. Dans l'exemple précédent, le message est la_méthode() et l'objet est un_objet.

La liste des arguments de la méthode permet de spécifier quelles informations on va passer à la méthode et quels sont leurs types.

Voici un exemple de méthode qui attend deux paramètres et qui renvoie la somme de ces deux paramètres :

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

1.3.6.) DROITS D'ACCES.

Un des grands avantages des classes est qu'on peut restreindre l'accès à certains de ces membres (variables et méthodes) par d'autres objets. Ceci s'appelle l'encapsulation. En Java, on spécifie les droits d'accès à l'aide de 5 mot-clés : private, protected, public et package :

Spécificateur	Classe	Classe descendantes	Package	Tous
private	X			
protected	X	X	X	
public	X	X	X	X
package	X		X	

Le mot clé « Private » fixe l'accès le plus restrictif. Un membre private n'est accessible qu'à partir de la classe dans lequel il est défini. La classe suivante contient une variable private et une méthode private :

```
class Alpha {
    private int iamprivate;
    private void privateMethod() {
        System.out.println("privateMethod");
    }
}
```

Le mode d'accès protected autorise l'accès à partir de la classe, de ses sous-classes et des classes se trouvant dans le même package.

Le mode d'accès public est le plus tolérant. Il autorise l'accès à toutes les classes. Il ne faut utiliser ce mode d'accès seulement si aucune manipulation indésirable peut être faite par un utilisateur de la

classe.

Le mode d'accès package est le mode par défaut. Il autorise l'accès à toutes les classes se trouvant dans le même package.

1.4.) main : point d'entrée du programme.

On lance l'exécution d'un programme Java en démarrant une machine virtuelle Java (la JVM) avec, en paramètre, le nom de la classe de démarrage, laquelle doit forcément; contenir une méthode *main*. Une fois que la JVM s'est mise en place, elle lance le programme proprement dit, par la première instruction de la méthode main, et ce, sans instancier d'objet à partir de la classe de démarrage. Cela fonctionne, car la méthode est déclarée **static** : c'est à dire qu'elle existe sans qu'il y ait eut instanciation. La tâche principale de cette méthode est alors d'instancier des objets sur différentes classes afin que le programme puisse travailler.
Une chose importante doit bien être comprise : comme la méthode *main* existe indépendamment de toute classe, si elle doit utiliser des attributs ou des méthodes de la classe, il faut alors que ces champs soient eux aussi déclarés **static**, sans quoi, ils n'existent pas. Plus formellement, les méthodes déclarées statiques, sur une classe, ne peuvent en manipuler que des champs statiques.
Notons au passage que la méthode main admet en paramètre un tableau de chaînes de caractères ("`String argv[]`"). Celui-ci contient les éventuelles options spécifiées sur la ligne de commande, lors du lancement de la JVM. Pour connaître la taille du tableau, il suffit bien entendu de faire « argv.length ». A titre d'information, le nom du paramètre peut-être n'importe quel nom, mais il est de bon ton d'utiliser *argv*.

```
class Start {
  static int a = 3;
  static public void main(String argv[]){
    a += 5;
    System.out.println("a^2 = " + Square(a));
  }
  static int Square(int value){
    return value*value;
  }
}
```

Une conséquence logique d'une définition statique est la suivante : les champs statique d'une classe sont partagés par toutes les instances de cette classe: En effet, comme tout champs statique existe indépendamment de toute instanciation d'objet, il existe aussi après une quelconque instanciation. L'exemple suivant reprend ce qui vient d'être dit : attention tout de même, car il peut dérouter.

```
class Start {
  static int a = 3;
  static public void main(String argv[]){
    Start s1=new Start(), s2=new Start();
    s1.a++;   s2.a++;
    System.out.println("a = " + Start.a);
  }
}

>java Start
5
>
```

Le programme se lance en exécutant la méthode *main* statique. Celle-ci instancie deux objets à partir

de la classe *Start* elle-même (ca marche parfaitement). On pourrait, dans le *main*, si on le voulait, faire "`s1.main()`" qui instancierait encore deux objets, mais on aurait très rapidement un souci de mémoire (qu'en pensez vous ?). Ensuite, sur chacun des deux objets, on incrémente la variable *a*. Mais comme elle est partagée, on incrémente finalement la variable de deux unités. Finalement, on affiche le résultat : 5. Notez qu'on accède ici à la variable en utilisant le nom de la classe. Ceci est permit car cela permet d'accéder des champs statiques d'une classe à partir d'une autre.

1.5.) L'héritage.

L'extension (ou héritage) va permettre de factoriser le code. L'idée principale consiste à définir une classe à partir d'une autre. La classe définie à partie d'une autre sera nommée classe fille. Celle qui sert à définir une classe fille sera nommée classe mère. On dit alors que la classe fille hérite (ou dérive ou étend) de la classe mère. Une remarque importante peut déjà être faite : une classe fille dérive d'une **unique** classe mère, l'héritage multiple n'étant pas supporté par le langage Java. Une fois que l'héritage est spécifié, la classe fille possède aussi l'ensemble des attributs et des méthodes de sa classe mère.

Au niveau de la syntaxe à utiliser pour définir un lien d'héritage, c'est très simple. Il suffit d'ajouter le mot clé **extends** suivit du nom de la classe mère, de suite après le nom de la classe fille, et ce dans la définition de cette dernière.

Un point important et souvent source d'erreur est à éclaircir. On n'hérite en aucun cas des constructeurs. Si vous ne spécifiez pas explicitement un constructeur particulier, vous ne pourrez l'utiliser, ce même s'il en existe un défini dans la classe mère. Par contre, des règles existent sur l'utilisation des constructeurs de la classe mère dans les constructeurs d'une classe fille quelconque.

Avant de voir ces règles en détail, quelques précisions sont à apporter sur deux mots clés du langage : **super** et **this**. Le premier sert à accéder les définitions de classe au niveau de la classe parente de la classe considérée (ces définitions pouvant être soit des méthodes ou des constructeurs). Le second sert à accéder à la classe courante. L'exemple suivante devrait mieux vous faire comprendre les choses. Les règles suivent directement l'exemple.

"Classe1.java"	"Classe2.java"

`class Classe1 {`
`  Classe1(){`
`  System.out.println("Classe1");`
`  }`
`  Classe1(int val){`
`  this();`
`  System.out.println(val);`
`  }`
`}`
	`class Classe2 extends classe1 {`
	`  Classe2(){`
	`  super(5);`
	`  System.out.println("Classe2");`
	`  }`
	`  Classe2(int val){`
	`  System.out.println(val);`
	`  }`

exemple de crétion d'objet	résultats
new Classe1();	Classe1
new Classe1(3);	Classe1 3
new Classe2();	Classe1 5 Classe2
new Classe2(2);	Classe1 2

Règle 1 : si la première instruction d'un constructeur ne commence pas par le mot clé **super** le constructeur par défaut de la classe mère est appelé (faites attention à ce qu'il soit définit).

Règle 2 : un appel de constructeur de la classe mère peut uniquement se faire qu'en première instruction d'une définition de constructeur. Une conséquence évidente est qu'on ne peut utiliser qu'un seul appel au constructeur de la classe mère.

```
class Shape {
Point2D centre;
Shape(){ centre=new Point2D(); }
Shape(Point2D c){ centre=c; }
void Move(Vector2D vecteur) {
centre.x += vecteur.x; centre.y += vecteur.y; }
void Move(double x,double y) {
centre.x += x; centre.y += y; }
void MoveH(double x) { centre.x += x; }
void MoveV(double y) { centre.y += y; }
void  print(String  nature){  System.out.print("Objet  "+nature"
:\n\tcentre : "); centre.print(); }
void println(){ print("Shape"); System.out.println(""); }}

class Square extends Shape {
double longueur;
Square(){ longueur=1; }
Square(Point2D c,double l){ super(c); longueur=l; }
void println(){ super.print("Square");
System.out.println("\n\tlongueur : "+longueur); }}

class Circle extends Shape {
double rayon;
Circle(){ rayon=1; }
Circle
(Point2D c,double r){ super(c); rayon=r; }
void println(){ super.print("Circle");
System.out.println("\n\trayon : "+rayon); }}
```

Le premier point important qu'il faut absolument bien assimiler, est que l'héritage supprime, en grande partie, les redondances dans le code. En effet, une fois la hiérarchie de classes bien établie, on localise en un point unique les sections de code (celles-ci restantes à tout moment accessibles grâce au mot clé **super**).
Seconde chose importante, et ce à condition que la hiérarchie de classes ait été bien pensée, on peut très facilement rajouter, après coup, une classe, et ce à moindre coup, étant donné que l'on peut réutiliser le code des classes parentes.
Dernier point, si vous n'aviez pas encore modélisé un comportement dans une classe donnée, et que vous vouliez maintenant le rajouter, une fois l'opération terminée, ce comportement sera alors directement utilisable dans l'ensemble des sous-classes de celle considérée.

2.) La notation unifiée « UML ».

UML (Unified Modeling Language, traduisez « langage de modélisation objet unifié ») est né de la fusion des trois méthodes qui ont le plus influencé la modélisation objet au milieu des années 90 : OMT, Booch et OOSE.

Issu « du terrain » et fruit d'un travail d'experts reconnus, UML est le résultat d'un large consensus. De très nombreux acteurs industriels de renom ont adopté UML et participent à son développement.

UML est avant tout un support de communication performant, qui facilite la représentation et la compréhension de solutions objet :

- Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.
- L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en font un langage universel.

UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles.

UML définit 9 diagrammes pour représenter les différents points de vue de la modélisation. Ils permettent de visualiser et de manipuler les éléments de la modélisation. Les diagrammes définis par UML sont les suivants :

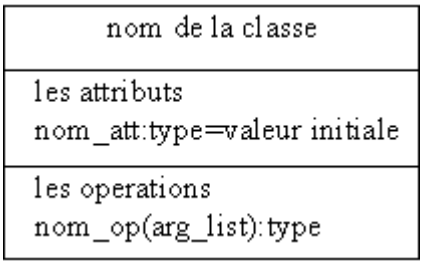
- **Les diagrammes d'activité** : représentation du comportement d'une opération en terme d'action
- **Les diagrammes de cas d'utilisation** : représentation des fonctions du système du point de vue de l'utilisateur.
- **Les diagrammes de classes** : représentation de la structure statique en terme de classes et de relations.
- **Les diagrammes de collaboration** : représentation spatiale des objets, des liens et des interactions.
- **Les diagrammes de déploiement** : représentation du déploiement des composants sur les dispositifs matériels.
- **Les diagrammes d'états-transitions** : représentation du comportement d'une classe en terme d'état.
- **Les diagrammes d'objet** : représentation des objets et de leurs relations, correspond à un diagramme de collaboration simplifié, sans représentation des envois de message
- **Les diagrammes de séquence** : représentation temporelle des objets et de leurs interactions.

2.1.) Les diagrammes de classes.

Les diagrammes de classes sont la représentation de la structure statique en terme de classes de relations. Les objets sont les instances des classes et les liens les instances des relations

2.1.1.) LES CLASSES.

Les classes sont représentées par un rectangle avec trois compartiments :



La déclaration d'un attribut a la syntaxe suivante :

```
Nom_Attribut : Type_attribut = Valeur_Initiale
```

UML permet de définir les attributs dérivés (attributs qui peuvent être calculés par d'autres éléments déjà existants). Ces attributs sont marqués par le symbole " / ". Par exemple, pour une classe rectangle, on connaît les attributs longueur et largeur. L'attribut dérivé serait la surface.

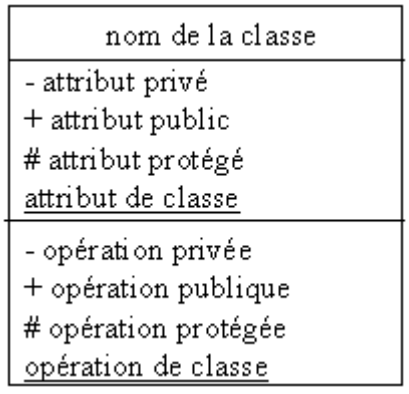
La déclaration d'une opération a la syntaxe suivante :

```
Nom_Opération ( Nom_Argument : Type_Argument = Valeur_par_defaut...)
: type_Retourné
```

UML définit trois niveaux de visibilité des attributs et des opérations :

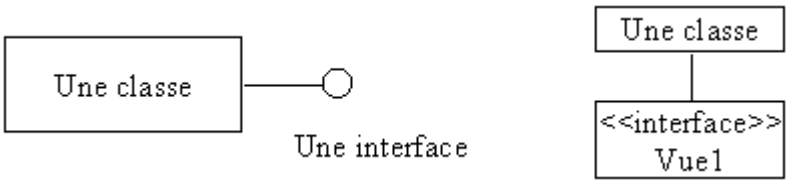
- **-** privé (l'élément n'est visible que dans la classe)
- **+** public (l'élément est visible par toutes les autres classes)
- **#** protégé (visible par la classe et ses sous classes)

La notation UML prévoie aussi les variables et les opérations de classes. Ces éléments sont identifiés par leur nom souligné dans la représentation de la classe.



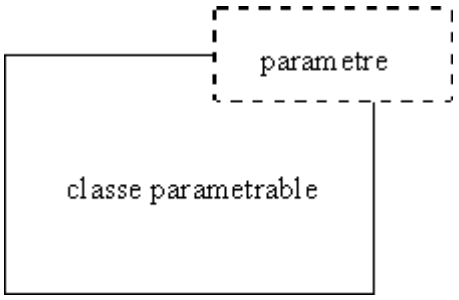
2.1.2.) LES INTERFACES DE CLASSES.

UML permet de représenter les interfaces des classes (qui décrit le comportement visible de la classe) au moyen d'un cercle qui leur est associé ou par une classe avec le stéréotype <<interface>>.



2.1.3.) LES CLASSES PARAMETRABLES.

Les classes paramétrables sont des modèles de classe. Ces dernières doivent être instanciées avec des paramètres pour obtenir une classe concrète. UML représente une classe paramétrable de la façon suivante :



Les types de classes particulières sont définis par des stéréotypes. Par exemple :
<< utilitaire >> Ce genre de classe permet de regrouper des éléments sans vouloir construire une classe complète (ex : des fonctions mathématiques).

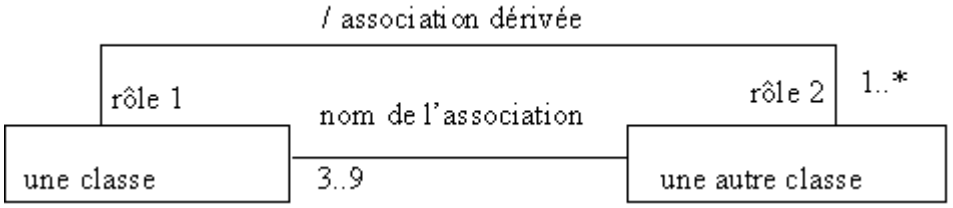
2.1.4.) LES ASSOCIATIONS.

Les associations en UML sont représentées par des traits continus entre les classes. Pour plus de renseignements, il est utile de nommer les associations. Pour cela, le nom de ces dernières se met au milieu de la ligne qui symbolise l'association, en italique sous forme verbale en général. Le sens de lecture de l'association peut être précisé au moyen des symboles " < " et " > ". De même, pour plus de compréhension de l'association, on peut indiquer à chaque extrémité le rôle qui décrit comment une classe voit une autre classe au travers d'une association. Le nommage des rôles est sous forme nominale.

On peut aussi préciser la multiplicité des associations. Ceci indique combien d'objets de la classe considérée participent à la relation et peuvent être liés avec un objet de l'autre classe. UML définit la syntaxe suivante :

1	*un et un seul*
0..1	*zéro ou un*
M .. N	*de M à N*
*****	*plusieurs*
0..*	*de zéro à plusieurs*
1..*	*de un à plusieurs*

Exemple :

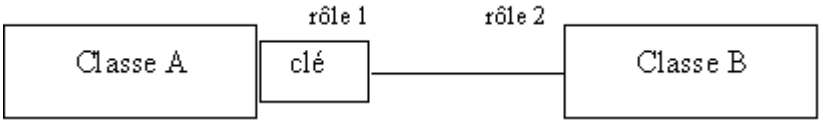


2.1.5.) LES CONTRAINTES.

Pour les associations, on peut introduire des contraintes. Une contrainte est une relation sémantique entre des éléments du modèle qui spécifie les conditions et les propositions qui doivent être respectées. Certains types de contraintes (comme une contrainte d'association "ou") sont prédéfinis dans UML, les autres devant être définis par les utilisateurs (ex : ordonnée). Une contrainte est représentée par un texte entre crochets ({}). UML ne prescrit pas le langage dans lequel la contrainte doit être écrite.

2.1.6.) LES RESTRICTIONS.

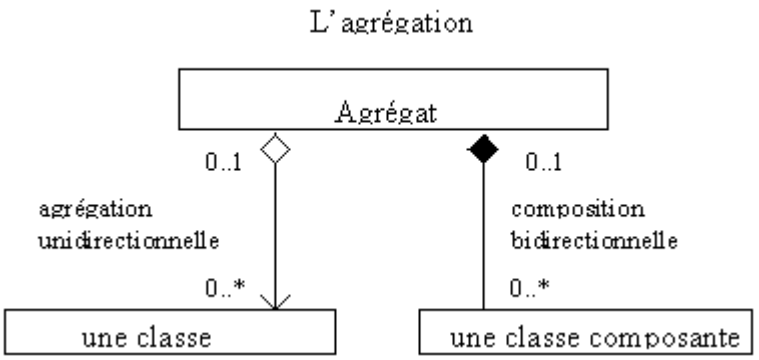
La restriction des associations est possible avec la notation UML. La restriction consiste à sélectionner un sous ensemble d'objets qui participent à l'association. La sélection des objets est caractérisée par un attribut que l'on appelle clé. Tous les objets de la classe qui vérifient la clé feront partie de l'association.



2.1.7.) L'AGREGATION.

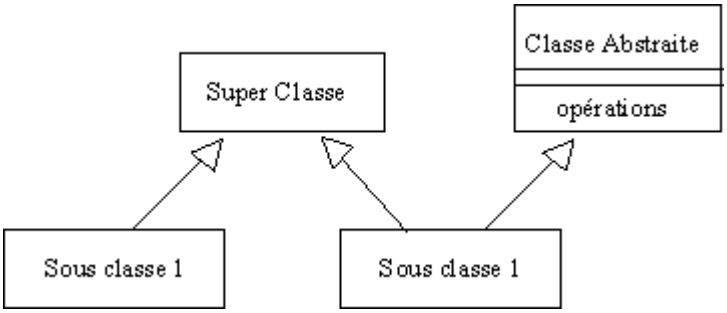
Une agrégation représente une association non symétrique dans laquelle une extrémité de l'association joue un rôle prédominant par rapport à l'autre extrémité. (par exemple, une classe qui fait partie d'une autre, une action d'une classe impliquant une action dans une autre classe...). L'agrégation en UML se représente par un petit losange blanc du côté de l'agrégat.

Il existe un cas particulier de l'agrégation. On l'appelle la composition ; Il s'agit de l'association correspondant à la contenance. UML représente la composition par un losange noir



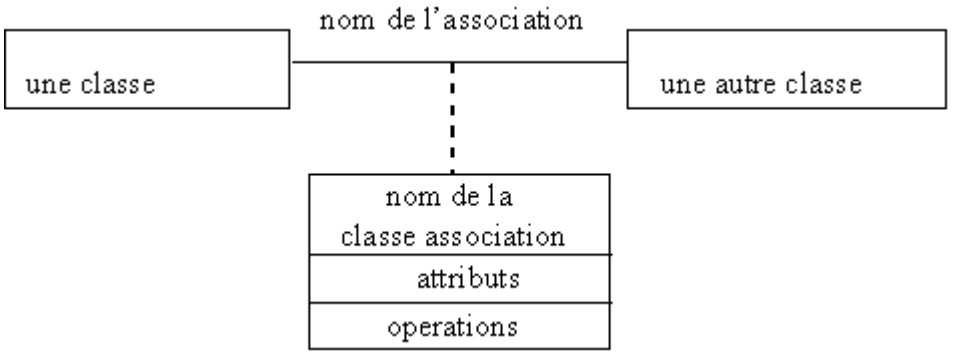
2.1.8.) LA GENERALISATION

La généralisation est représentée par une flèche qui pointe de la classe la plus spécialisée vers la classe plus générale. Elle exprime le fait que les éléments d'une classe soit décrits dans une autre classe. Les attributs, opérations et associations sont hérités par les sous classe. La tête de la flèche est caractérisée par un triangle vide.



2.1.9.) CLASSE ASSOCIATION.

UML définie les classes associations qui ont pour but d'ajouter des attributs et des opérations à l'association. UML représente la classe associative par une classe reliée par un trait en pointillé sur l'association concernée.



2.1.10.) CLASSE ABSTRAITE.

UML représente les classes abstraites par une classe dont le nom est " classe abstraite " et est notée en italique. Les classes abstraites ne sont pas instanciable. Elles servent de spécification

3.) Etude de la classe Object.

La classe Object se trouve dans le package java.lang.

Elle n'hérite d'aucune classe. Par contre, toutes les autres classes héritent de Object. Il s'agit donc de la racine de toute hiérarchie de classes. Donc, tous les objets (y compris les tableaux) implémentent les méthodes de cette classe.

Il s'agit d'une classe publique.

Ce n'est évidement pas une classe finale.

Constructeurs	
Object( )	

Methodes	
`protected Object`	clone( ) Créé une copie de l'objet..
`boolean`	equals(Object `obj`) Indique si l'objet est égal à celui passé en paramètre.
`protected void`	finalize( ) Cette méthode est appellée par le "Garbage Collector" avant la destruction de l'objet.
`Class`	getClass( ) Renvoie la classe de l'objet.
`int`	hashCode( ) Retourne un "hashcode" pour cet objet.
`void`	notify( ) Réveille un processus qui attend sur la disponibilité de cet objet.
`void`	notifyAll( ) Réveille tout les processus en attente.
`String`	toString( ) Retourne une représentation en chaîne de cette objet.
`void`	wait( ) Demande au processus courant d'attentre jusqu'à ce qu'un autre processus appelle la méthode notify() ou notifyall()
`void`	wait(`long timeout`) Demande au processus courant d'attentre jusqu'à ce qu'un autre processus appelle la méthode notify() ou notifyall() ou jusqu'à ce que le timeout soit dépassé.
`void`	wait(`long timeout, int nanos`) Demande au processus courant d'attentre jusqu'à ce qu'un autre processus appelle la méthode notify() ou notifyall() ou jusqu'à ce que le timeout soit dépassé.

4.) Etude de la classe Class.

La classe Class se trouve dans le package java.lang.

Elle n'hérite de la classe Objet.

Il s'agit d'une classe publique.

Il s'agit d'une classe finale donc non héritable.

Methodes	
`static Class`	forName(String `className`) Retourne la classe de nom className.
`static Class`	forName(String `name, boolean initialize,` ClassLoader `loader`) Renvoie la classe de nom className en utilisant ClassLoader.
`Class[]`	getClasses() Renvoie un tableau contenant toutes les classes publiques et toutes les interfaces membres de l'objet classe.
`ClassLoader`	getClassLoader() Renvoie l'objet ClassLoader de l'objet classe.
`Constructor`	getConstructor(Class`[] parameterTypes`) Renvoie un objet constructeur public qui attend les paramétres spécifiés dans `parameterTypes`.
`Constructor[]`	getConstructors() Renvoie un tableau contenant tous les constructeurs publics.
`Class[]`	getDeclaredClasses() Retourne un tableau contenant toutes les classes et interfaces déclarées comme menbres de cette classe.
`Constructor`	getDeclaredConstructor(Class`[] parameterTypes`) Renvoie un objet constructeur qui attend les paramétres spécifiés dans `parameterTypes`.
`Constructor[]`	getDeclaredConstructors() Renvoie un tableau contenant tous les constructeurs.
`Field`	getDeclaredField(String `name`) Retourne un objet Field représenté par name.
`Field[]`	getDeclaredFields() Renvoie un tableau des objets Field déclaré dans cette classe.
`Method`	getDeclaredMethod(String `name,` Class`[] parameterTypes`) Retourne une méthode de nom name et acceptant les paramètres `parameterTypes.`
`Method[]`	getDeclaredMethods() Renvoie un tableau des méthodes déclarées dans cette classe.
`Field`	getField(String `name`) Retourne (s'il existe) le champ public de nom name.
`Field[]`	getFields() Retourne un tableau contenant tout les objets Field accessibles.
`Class[]`	getInterfaces() Renvoie les interfaces implémentés par la classe.
`Method[]`	getMethods() Retourne un tableau des méthodes de cette classe.
`int`	getModifiers() Retourne le modificateur Java pour cette classe.

`String`	getName() Retourne le nom de la classe.
`Package`	getPackage() Renvoie le package de cette classe.
`Class`	getSuperclass() Retourne la classe ancêtre de la classe.
`boolean`	isArray() Vrai si la classe est un tableau.
`boolean`	isInstance(Object `obj`) Renvoie vrai si obj est une instance de la classe.
`boolean`	isPrimitive() Détermine si cette classe est un type primitif.
`Object`	newInstance() Créé une nouvelle instance de cette classe.
`String`	toString() Conversion en chaîne.

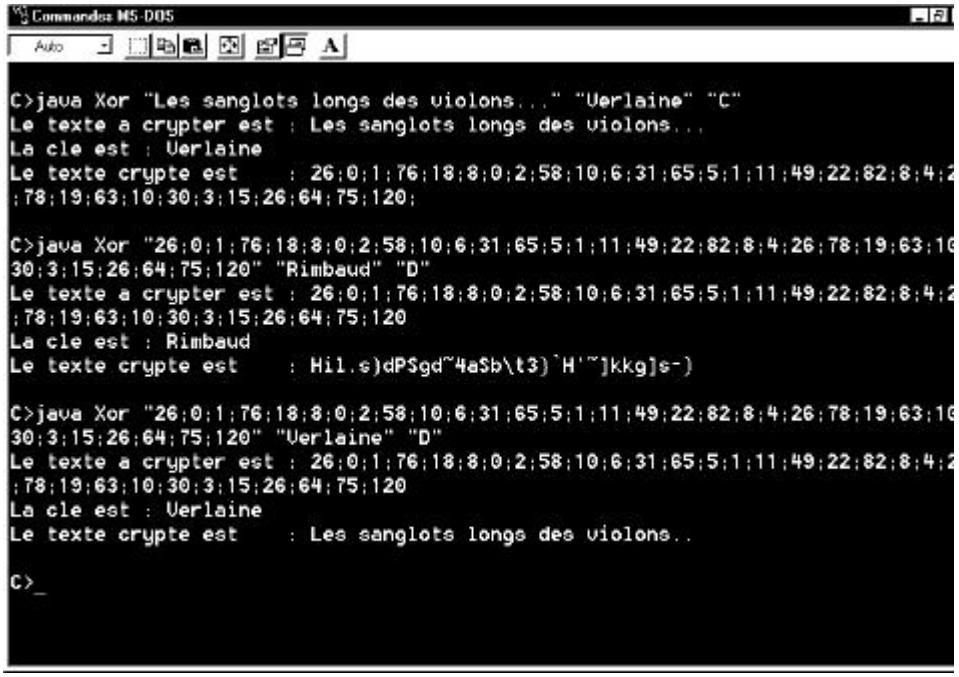
5.) La cryptologie.

5.1.) "Ou exclusif" simple.

Le changement d'échelle que constitue le passage du niveau de codage "caractère" au niveau de codage "bit" a permis de manipuler véritablement les nombres, en enrichissant la cryptographie de techniques mathématiques de brouillage non disponibles au niveau caractère.

La technique la plus simple est d'appliquer un "ou exclusif" entre le texte à chiffrer et la clé. Le "ou exclusif" se définit de la manière suivante :

	0	**1**
0	0	1
1	1	0



```
public class Xor {

public static void main(String[] argv) {
   String chaine_a_crypter = " ";
   String cle = " ";
   String chaine_cryptee = new String();
   int tableau_cle [];
   char c;
   int pos,pos_cle;
   boolean decodage = false;

   if (argv.length > 0)
        chaine_a_crypter = argv[0]; else chaine_a_crypter = "La
reine blanche comme lys";
   if (argv.length > 1)
       cle = argv[1]; else cle = "FrancoisVillon";
   if (argv.length > 1)
   {
        c = argv[2].charAt(0);
        if ((c=='d') | (c=='D')) decodage = true;
   }

   // On place les octets correspondants à la clé dans un
tableau
   tableau_cle = new int[cle.length()];
   for (int i=0;i<cle.length();i++)
   {
        c = cle.charAt(i);
        /*pos = ascii.indexOf(c);
        if (pos<0) pos = 0;
        tableau_cle[i] = pos + 32; */
        pos = (int)c;
        tableau_cle[i]=pos;

   }

   if (! decodage)
   {
        // Codage à l'aide d'un ou exclusif
        pos_cle = 0;
        for (int i=0;i<chaine_a_crypter.length();i++)
        {
             c = chaine_a_crypter.charAt(i);
             pos = (int)c;
             pos = pos ^ tableau_cle[pos_cle];
             chaine_cryptee = chaine_cryptee + pos + ";";
             pos_cle ++;
             if (pos_cle >=cle.length()) pos_cle = 0;
        }
   }
   else
   {
        // Décodage
        int i = 0;
        pos_cle = 0;
        String s = new String();
```

```
        while (i < chaine_a_crypter.length())
        {
             if (chaine_a_crypter.charAt(i) == ';')
             {
                  pos = Integer.parseInt(s);
                  pos = pos ^ tableau_cle[pos_cle];
                  c = (char)pos;
                  chaine_cryptee = chaine_cryptee + c;
                  pos_cle ++;
                  if (pos_cle >=cle.length()) pos_cle = 0;
                  s = new String();
             }
             else
             {
                  s = s + chaine_a_crypter.charAt(i);
             }
             i ++;
        }
   }
   // Affichage des résultats
   System.out.println("Le    texte    a    crypter    est    :
"+chaine_a_crypter);
   System.out.println("La cle est : "+cle);
   System.out.println("Le   texte   crypte   est            :
"+chaine_cryptee);
    }
}
```

5.2.) Le D.E.S.

5.2.1.) INTRODUCTION

Le D.E.S. (Data Encryption Standard, c'est-à-dire Standard de Chiffrement de Données) est un standard mondial depuis plus de 15 ans. Bien qu'il soit un peu vieillissant, il résiste toujours très bien à la cryptanalyse et reste très sûr contre les ennemis sauf peut-être les plus puissants.

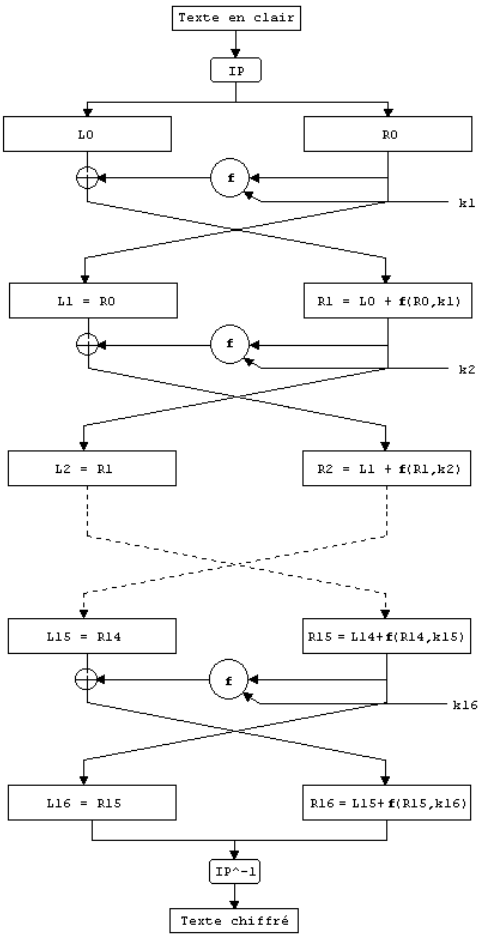
Au début des années 70, le développement des communications entre ordinateurs a nécessité la mise en place d'un standard de chiffrement de données pour limiter la prolifération d'algorithmes différents ne pouvant pas communiquer entre eux. Pour résoudre ce problème, L'Agence Nationale de Sécurité américaine (N.S.A.) a lancé des appels d'offres.

La société I.B.M. a développé alors un algorithme nommé Lucifer, relativement complexe et sophistiqué. Après quelques années de discussions et de modifications, cet algorithme, devenu alors D.E.S., fut adopté au niveau fédéral le 23 novembre 1976.

C'est, comme son nom l'indique, un standard ; pour cette raison l'algorithme est parfaitement défini et immuable. Toute modification apportée à D.E.S. impose de renommer l'algorithme obtenu.

5.2.2.) PRESENTATION DE L'ALGORITHME

Le D.E.S. est un système de chiffrement par blocs. Cela signifie que D.E.S. ne chiffre pas les données à la volée quand les caractères arrivent, mais il découpe virtuellement le texte en clair en blocs de 64 bits qu'il code séparément, puis il les concatène. Un bloc de 64 bits du texte clair entre par un coté de l'algorithme et un bloc de 64 bits de texte chiffré sort de l'autre coté.



C'est un algorithme de cryptage à clef secrète. La clef sert donc à la fois à crypter et à décrypter le message. Cette clef a ici une longueur de 64 bits, c'est-à-dire 8 caractères, mais seulement 56 bits sont utilisés. On peut donc éventuellement imaginer un programme testant l'intégrité de la clef en exploitant ces bits inutilisés comme bits de contrôle de parité.

L'algorithme est assez simple puisqu'il ne combine en fait que des permutations et des substitutions. On parle en cryptologie de techniques de confusion et de diffusion.

L'entière sécurité de l'algorithme repose sur les clefs puisque l'algorithme est parfaitement connu de tous. La clef de 64 bits est utilisée pour générer 16 autres clefs de 48 bits chacune qu'on utilisera lors de chacune des 16 itérations du D.E.S.. Ces clefs sont les mêmes quel que soit le bloc qu'on code dans un message. On les calcule donc une fois pour toute.

5.2.3.) GENERATION DES CLES

On commence par réduire les 64 bits de la clef DEF à 56 en ignorant un bit sur 8. Ceci est décrit dans le tableau suivant :

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Après que la clef de 56 bits ait été extraite, une clef de 48 bits différente est engendrée pour chaque ronde du DES. Ces clés Ki sont déterminées de la manière suivante : Tout d'abord, la clé de 56 bits est divisée en deux moitiés de 28 bits G0 et D0. Ensuite, les moitiés sont décalées vers la gauche d'une ou deux positions en fonction de la ronde. Le nombre de bits de décalage est donné par le tableau suivant :

Ronde	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Nombre de décalages	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

On obtient donc G1 et D1. On forme alors le bloc G1D1 (en concaténant les deux blocs bout à bout) dans lequel on sélectionne 48 bits, lesquels forment la clef K1. Comme cette opération combine une permutation des bits avec une sélection d'un sous-ensemble des bits, elle est appelée **permutation compressive**, ou encore **choix permuté**. Le tableau suivant défini la permutation compressive :

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Par exemple, le bit en position 14 de la clé décalée se retrouve en position 1 et le bit en position 9 est ignoré. Les décalages ont pour but de rendre différents les sous-ensembles de bits utilisés dans chaque sous-clé.

Ce calcul se généralise pour calculer les 15 autres clefs Ki à partir de Gi-1 et Di-1 à ceci près que le nombre de bits de décalage de Gi et de Di varie en fonction de i. On génère donc 16 clefs Ki où $i \in \{1;2;3;...;16\}$.

5.2.4.) L'ALGORITHME PROPREMENT DIT

Le bloc de 64 bits issu du message subi une permutation initiale P_0 qui renvoie les 64 bits dans un autre ordre. Le tableau suivant décrit la permutation initiale :

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

On coupe ensuite le bloc ainsi obtenu en 2 blocs de 32 bits L0 et R0. Le D.E.S. réalise ensuite 16 fois une série d'opérations sur ces 2 blocs : on appelle cela les 16 itération du D.E.S. ou les 16 rondes du D.E.S.. Chacune de ces rondes, nommée parfois fonction f, peut être décrite en 5 étapes; elle utilise le bloc L_i et le bloc R_i, ainsi que la clef K_i précédemment générée.

– Première étape :

Le bloc R_i subie une permutation expansive qui le fait passer de 32 bits à 48 bits en répétant certain bits. On obtient le bloc R_i'. Cette opération a deux buts : le résultat à la même taille que la clé et elle fourni un résultat plus long qui pourra être comprimé pendant l'opération de substitution. De plus - et surtout - elle permet à un bit d'affecter deux substitutions ce qui fait que la dépendance entre bits d'entrée et de sortie se dilue plus vite. On parle d'effet d'avalanche. Le tableau suivant montre la permutation expansive :

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

– Deuxième étape :

On réalise un OU-Exclusif entre les 48 bits de la clef K_i et les 48 bits du bloc R_i' que l'on vient de générer.

– Troisième étape :

Le bloc ainsi obtenu est coupé en 8 blocs de 6 bits chacun. Chaque bloc est substitué de la manière suivante :

Un bloc est représenté par les 6 bits le constituant : (b1, b2, b3, b4, b5, b6). A l'aide de ces bits on forme 2 nombres représentant une colonne et une ligne de la matrice de substitution du D.E.S. qui est

parfaitement connue et définie. Ainsi, en regroupant b1 et b6 on forme le nombre (b1b6) compris entre 0 et 3 qui donne l'indice de ligne ; le regroupement des autres bits forment le nombre (b2b3b4b5) compris entre 0 et 15 qui indique le numéro de colonne. Ces 2 indices fournissent la valeur correspondante dans la matrice de substitution. C'est un nombre compris entre 0 et 15 c'est-à-dire codé sur 4 bits. On substitue donc 4 bits à 6 bits. Pour améliorer un peu l'algorithme, chaque bloc utilise une matrice de substitution différente. Il y a donc 8 matrices de substitutions pour chacun des 8 blocs de 6 bits. En revanche les mêmes matrices sont utilisée dans 2 itérations différentes du D.E.S. sinon sa programmation serait délicate. Les 8 blocs de 4 bits ainsi obtenus sont ensuite concaténés.
Le résultat de la troisième étape est donc la substitution d'un bloc de 48 bits par un bloc de 32 bits obtenu grâce au tables de substitution. Les tables de substitutions sont décrites ci-après :

Table S 1 :															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
Table S 2 :															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
Table S 3 :															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
Table S 4 :															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
Table S 5 :															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
Table S 6 :															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
Table S 7 :															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
Table S 8 :															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

– Quatrième étape :

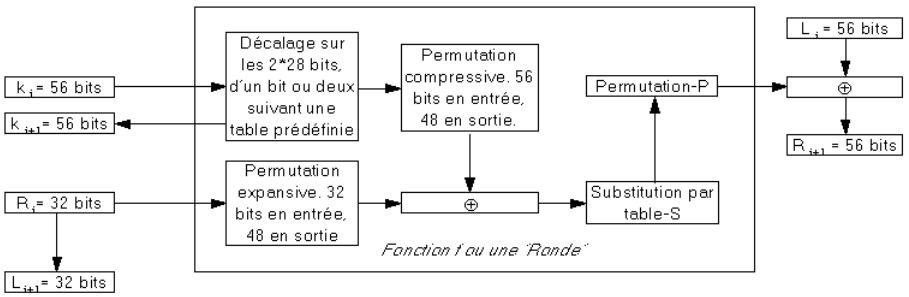
Les 32 bits renvoyés par l'étape 3 sont permutés par la permutation P qui renvoie 32 bits. On obtient le bloc R_i''. La permutation P est décrite dans le tableau suivant :

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

– Cinquième étape :

On réalise un OU-Exclusif entre le bloc Li et le bloc R_i'' qui ont bien tous les deux 32 bits. On obtient alors le bloc R_{i+1}. Le bloc L_{i+1} est pris égal au bloc R_i du début de la première étape. Cette dernière étape renvoie donc les blocs L_{i+1} et R_{i+1} qui seront utilisés pour la ronde suivante.
On a donc $L_{i+1} = R_i$ et $R_{i+1} = f(L_i, R_i, K_i)$.
Après les 16 rondes, on concatène les blocs R_{16} et L_{16} pour former le bloc (R_{16}, L_{16}) – notons que les blocs sont échangés : on ne forme pas le bloc (L_{16}, R_{16}) comme il aurait été logique de le faire ; ceci assure le caractère involutif de l'algorithme – auquel on applique la permutation P_{0-1} qui est la réciproque de la permutation initiale. On obtient – enfin ! – le bloc chiffré qui fait donc 64 bits.

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25



5.2.5.) DECHIFFREMENT DU D.E.S.

Bien que toutes les opérations réalisées semblent à sens unique, il n'en est rien en fait puisque toutes les opérations effectuées sont bijectives. Le même algorithme pourra donc être utilisé pour le déchiffrement la seule différence étant l'ordre d'utilisation des clefs : il faut prendre les clefs dans l'autre sens c'est-à-dire en commençant par K_{16}. En revanche l'algorithme sera lu dans le même sens que lors du chiffrement.

5.2.6.) AVANTAGES ET INCONVENIENTS DU D.E.S.

Le D.E.S. possède 2 avantages de taille qui font de lui le système cryptographique le plus utilisé à l'heure actuelle qui sont sa sécurité et sa relative vitesse de chiffrement par des puces spécialisées.

En effet la sécurité du D.E.S. avec ses 16 rondes est grande et résiste à l'heure actuelle à toutes les attaques linéaires, différentielles ou par clefs corrélées, effectuées avec des moyens financiers et temporels raisonnables (i.e. moins de 10 millions de dollars et moins d'un mois). La grande sécurité repose sur ses tables de substitutions non linéaires très efficaces pour diluer les informations. De plus le nombre de clefs est élevé ($2^{56}=7{,}2*10^{16}$) et peut être facilement augmenté en changeant le nombre de bits pris en compte. D'autres avantages plus techniques au niveau cryptanalyse existent.

De plus, cet algorithme est relativement facile à réaliser matériellement et certaines puces chiffrent jusqu'à 1 Go de données par seconde ce qui est énorme : c'est plus que ce qu'est capable de lire un disque dur normal. Pour les industriels c'est un point important notamment face à R.S.A.

Mais tout n'est pas quand même rose pour notre petit algorithme : il a quand même quelques défauts. Le principal est le sentiment de frustration de beaucoup sur sa réelle sécurité. La N.S.A. a-t-elle inséré une brèche secrète dans l'algorithme pour pouvoir contrôler les messages chiffrés ? Mais les nombreuses études développées depuis n'ont jamais rien prouvé. De plus on a constaté l'existence de certaines clefs dites faibles c'est-à-dire dont l'utilisation peut permettre un décodage plus facile qu'avec d'autres clefs. Il faut donc bien choisir sa clef si on veut une réelle efficacité. Enfin la polémique sur la longueur de la clef se poursuit afin de déterminer si une clef de 56 bits est assez longue. Il faut aussi noter que depuis peu, le gouvernement des Etats-Unis autorise l'exportation des systèmes cryptographiques ayant des clés inférieures ou égales à 56 bits. On est donc certain que le DES ne pose pas le moindre problème à la NSA quant à sa cryptanalyse. Aujourd'hui, casser le DES n'est donc pas à la portée du grand public, mais l'est pour un cryptographe et les entreprises (on se place bien sûr dans le cas où la clé utilisée est choisie aléatoirement)[1].

Toujours est-il que le D.E.S. (ou une de ses nombreuses variantes qui existent) a encore une belle perspective de vie avant que soit trouvé quelque chose de vraiment plus performant.

5.2.7.) VARIANTES DU D.E.S.

Des DES ont été modifiés avec d'autres tables-S, qui résistent cette fois complètement à la cryptanalyse différentielle et linéaire. C'est le RDES-1 à 4 (attention, le RDES tout court n'est pas sûr), et le SnDES. On ne peut pas éviter l'attaque exhaustive en modifiant les tables-S. La seule possibilité de l'éviter est de modifier la taille de la clé.

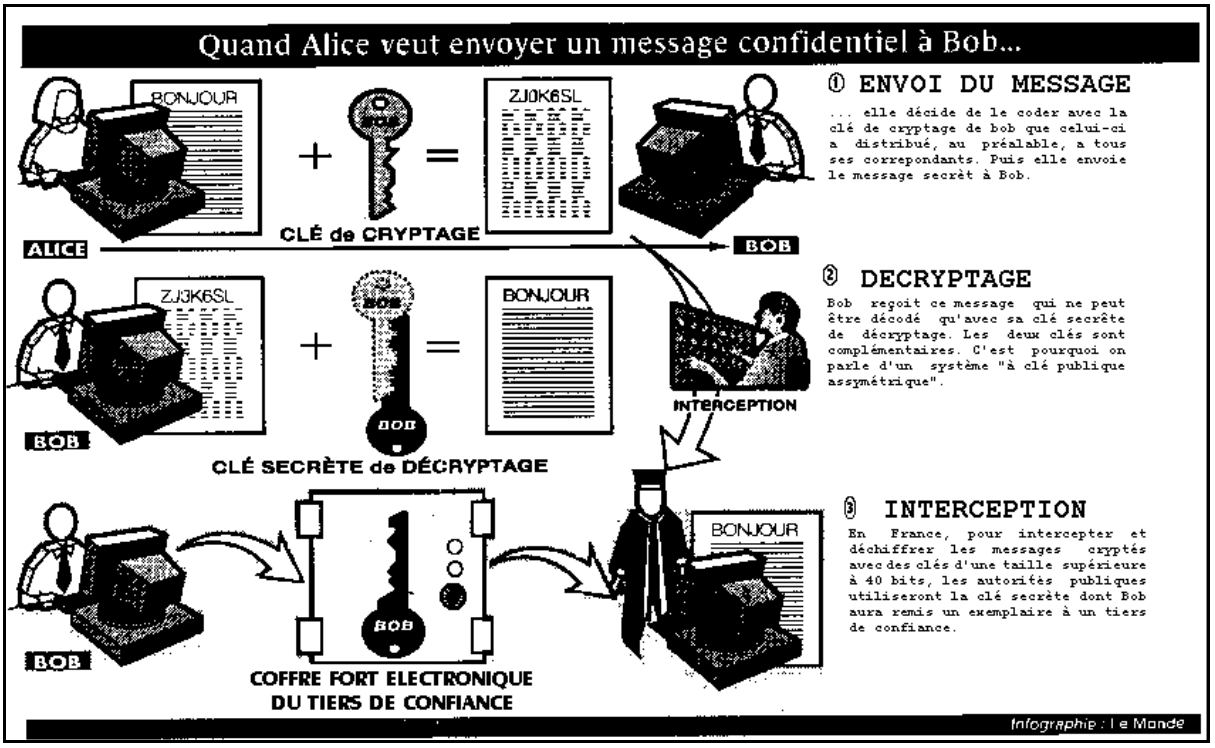
Il existe une version du DES assez connue sous le nom de Triple DES, noté aussi 3DES, car il utilise une clé de longueur triple en entrée, soit de 3*64 = 192 bits (24 caractères). Il y a toujours 1 bit sur 8 qui est exclut. La clé utilisée en interne par l'algorithme ne fait donc non pas 3*64 bits mais 3*56 = 168 bits. C'est cette dernière valeur qui compte.

Le 3DES : On découpe la triple clé en 3 clés de 64 bits. Pour chiffrer un texte, on chiffre avec la 1ère, on déchiffre avec la 2nd, et on rechiffre avec a $3^{ième}$. La programmation est simple, et le résultat s'en retrouve grandement augmenté : on peut démontrer que l'attaque par recherche exhaustive est de 2^111 clés (et non pas 2^168, comme on pourrait le croire), contrairement aux 2^56 clés du DES officiel.

Il existe encore d'autres variantes, comme les 2DES ou le GDES qui opère sur des blocs de textes variables etc...

[1] En 1981, Diffie et Hellman démontrèrent que retrouver la clé en 2 jours, par attaque exhaustive (qui essaie toutes les combinaisons de clés possibles) avec un ordinateur à architecture parallèle coûtait 50 millions de dollars. Ils admettaient que c'était hors de portées de quiconque, excepté des organisations comme la NSA. Ils considéraient aussi que d'ici 1990, le DES ne serait plus sûr du tout. En 1984, les puces étaient capables de faire 256000 chiffrements de bloc par secondes. En 1987, c'était plus d'un million, en 1995, on trouvait dans le commerce des puces chiffrant 8 millions de blocs / s. Les ordinateurs n'ont pas de si bon taux : Pc 486 33 Mhz = 40 600 blocs /s, Hp9000 = 196 000 blocs / s . En 1993, Michael Wiener a conçu les plans d'une machine coûtant 1 million de dollars, pouvant accomplir une attaque exhaustive (tester les 7*10^16 de clés possibles) en 3.5 heures .La machine n'a, on le suppose, jamais été fabriquée. Aujourd'hui en 1997, les plans de cette machine sont sur l'Internet. La machine revient à 100 000 dollars, et casse n'importe quelle clé DES en 35 heures ou moins.

5.3.) Le cryptage à clef publique.



5.4.) R.S.A

Les avantages offerts par le cryptage à clef publique entraîna tout naturellement la réalisation d'algorithmes suivant ce procédé. Si celui à empilement, de **Merkle** fut le premier du genre, le **RSA** (du nom de ses inventeurs Ron **Rivest**, Adi **Shamir** et Leonard **Adleman**, et datant de **1978)** parut peu après. Son indéniable popularité provient de sa simplicité (du point de vue de sa réalisation comme de sa compréhension). Par ailleurs, même si sa sécurité n'est pas théoriquement démontrée, RSA résiste pour l'instant à toute les tentatives de craquage, ce qui suggère un certain niveau de confiance dans l'algorithme.
En fait, les clefs publiques et privées sont générées à partir de deux grands nombres premiers (plus de 100 chiffres). Retrouver le texte en clair à partir d'une des clefs est équivalent à la factorisation du produit des deux nombres premiers. Par conséquent le niveau de sécurité de RSA dépend directement de la difficulté de factoriser des grands nombres.

5.4.1.) LA GENERATION DES CLEFS.

La première étape consiste à choisir deux nombres premiers p et q et d'en calculer le produit n :

$$n = p * q$$

On calcule ensuite le nombre φ égal à $\varphi = (p-1)*(q-1)$

On choisit ensuite la clé de chiffrement e telle que e n'ait pas de diviseur commun avec φ.

La clé de déchiffrement est le nombre d qui vérifie :

$$(e*d) \mod j = 1$$

d et e sont ainsi premiers entre eux.

Les nombres e et n forment la clef publique, le nombre d est la clef privée. p et q ne servent plus : ils peuvent être jetés, mais pas révélés.

Par exemple, si on choisit p = 11 et q=17 :

- n = 11*17 = **187.**
- φ= 10*16 = **160**.
- Les diviseurs de 160 étant 2,4,5,8,10,16,20,32,40 et 80, on choisit un nombre e n'étant pas divisible par 2,4,5,8,10,16,20,32,40 ou 80. On peut choisir e=**7**.
- La clé de déchiffrement d est le nombre tel que (e*d) mod φ = 1. On peut prendre **23** car (23*7) mod φ = 1.[2]

Le chiffrement du mot CERBERE pourrait se faire ainsi :

On associe à chaque lettre un nombre (par exemple sa position dans l'alphabet).

C	E	R	B	E	R	E
3	5	18	2	5	18	5

Chaque lettre est codée ainsi : code = valeur^e mod n, c'est à dire valeur^7 mod 187. On aura :

C	E	R	B	E	R	E
130	146	171	128	146	171	146

Pour le déchiffrement on calculera : valeur = code^d mod n, c'est à dire code^{23} mod 187 On aura :

C	E	R	B	E	R	E
3	5	18	2	5	18	5

5.4.2.) CHIFFREMENT ET DECHIFFREMENT D'UN MESSAGE

Dans la pratique, on commence par découper le message m à crypter en blocs numériques ayant chacun une représentation unique modulo n. Ainsi, si p et q sont tous deux des nombres premiers de 100 chiffres, n aura tout juste moins de 200 chiffres et chaque bloc de message mi doit avoir juste moins de 200 chiffres. Le message chiffré c sera constitué de manière similaire de blocs ci d'à peu près la même longueur.

La formule de chiffrement est simplement :

$$c_i = m_i^{\,e} \mod n$$

Pour déchiffrer un message, prenons chaque bloc ci et calculons :

$$m_i = c_i^{\,d} \mod n$$

[2] Ce nombre peut être calculé à l'aide de l'algorithme d'Euclide étendu. Voir page 34.

Cette formule permet de retrouver le message de départ. En effet (toutes les opérations sont effectuées modulo n) :

ci d = (mi e) d = mi ed = mi k (p - 1) (q - 1) + 1 = mi * mi k (p - 1) (q - 1) = mi*1 = mi

Le message aurait évidemment pu être chiffré avec d et déchiffré avec e.

Afin de clarifier les idées voici un exemple simple. On cherche à crypter la phrase suivante :

En cette foi je veux vivre et mourir.

On commence par convertir chaque caractère de la chaîne en son équivalent Ascii.

E	n		c	e	t	t	e		f	o	i		j	e		v	e	u	x	
069	110	032	099	101	116	116	101	032	102	111	105	032	106	101	032	118	101	117	120	032

v	i	v	r	e		e	t		m	o	u	r	i	r	.				
118	105	118	114	101	032	101	116	032	109	111	117	114	105	114	046	032	032		

On obtient donc le message:

```
069110032099101116116101032102111105032106101032118101117120032118105
1181141010321011160321091111171141051140460320 32
```

Prenons p = 2370331141et q = 3641786851, alors n = p*q = 8632240781809626991.

La clef de chiffrement e ne doit pas avoir de facteurs communs avec : (p - 1)*(q - 1) = 2370331140*3641786850 = 8632240775797509000.

Choisissons e aléatoirement, ... disons 4058653531. Dans ce cas d = 2032676768617160371[3].

Il nous reste alors à publier e et n, à oublier p et q, et à garder d secret.

Pour chiffrer le message m = `069110032099101116116101032102111105032106101032118101117120032118105118114101032101116032109111117114105114046032032` on procède tout d'abord à son découpage en blocs de 15 chiffres. On obtient :

```
069110032099101;116116101032102;111105032106101;032118101117120;0321
18105118114;101032101116032;109111117114105;114046032032;
```

Le premier bloc est chiffré par : $069110032099101^{4058653531} \bmod 8632240781809626991 = 1891393215034780648$.

En effectuant la même opération pour tous les blocs, on obtient la suite c =
```
1891393215034780648;4000749984415332053;4238183656635775304;85986749
6076890746;3728751979654969900;563432397904671210;621185868026245930
8;2404453770395775358;5246566146929389441;
```

Le déchiffrement s'effectue de la même manière, mais en utilisant la clef de déchiffrement. Donc :

$1891393215034780648^{2032676768617160371} \bmod 8632240781809626991 =$ 69110032099101. soit « *En ce* ».

[3] Ce nombre peut être calculé à l'aide de l'algorithme d'Euclide étendu. Voir page 34.

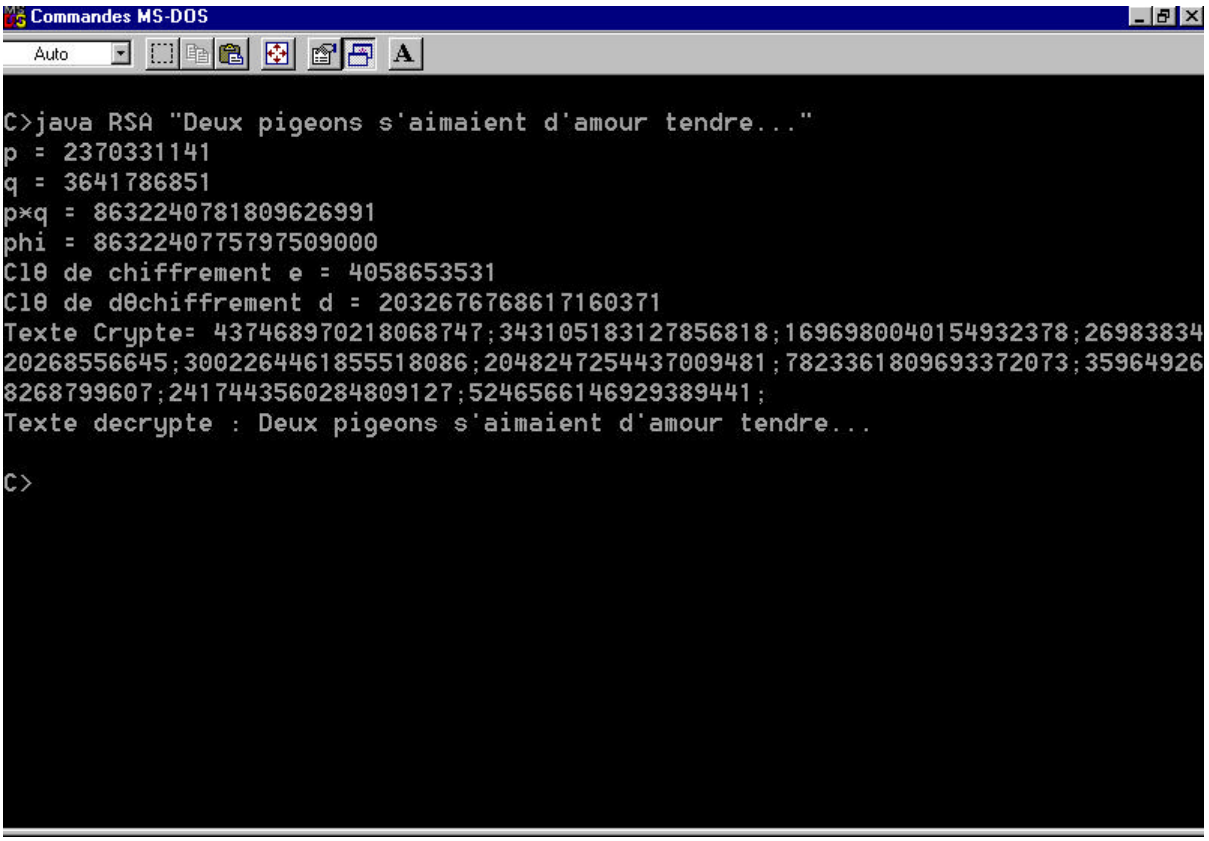
Le reste du message s'obtient par la même méthode.

5.4.3.) PROGRAMMATION

Afin de programmer le RSA d'une manière efficace, il est nécessaire d'utiliser des algorithmes suffisamment puissants pour éviter à la machine de trop gros calculs. Ainsi, il est clair que la fonction de mise à la puissance modulo n ne saurait consister en la mise à la puissance, puis au calcul du résultat modulo n.

Par ailleurs la programmation d'un algorithme tel que RSA n'a d'intérêt que dans la mesure où les clefs utilisées sont suffisamment grandes (typiquement une centaine de chiffre), sans quoi le décryptage devient trop facile. Par conséquent, il est indispensable de s'attacher à la réalisation de fonctions de création de clefs. L'exécution de cette tâche sera assurée, d'une part par une fonction de test probabiliste du caractère premier d'un nombre (pour trouver p et q), d'autre part par une fonction de calcul d'un l'inverse modulo n (pour trouver la clef de déchiffrement).

La classe BigInteger du langage Java fourni toutes les méthodes nécessaires à ce calcul.



```
import java.math.*;
import java.util.*;
class RSA {
public static void main(String[] argv)
{
   Random r = new Random(21);
   BigInteger p = new BigInteger(32, 2000, r);
   BigInteger q = new BigInteger(32, 2000, r);
   BigInteger pq = p.multiply(q);
   BigInteger e = new BigInteger(32,r);
   BigInteger phi;
   BigInteger d;
   String chaine_a_crypter;
   String chaine_cryptee = "";
   String s;
   if (argv.length > 0)
   chaine_a_crypter = argv[0]; else chaine_a_crypter = "En cette
foi je veux vivre et mourrir.";
   chaine_a_crypter = chaine_a_crypter + "          ";
   System.out.println("p = "+p);
   System.out.println("q = "+q);
   System.out.println("p*q = "+pq);
   phi =
(p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE)
);
   System.out.println("phi = "+phi);
   while (e.gcd(phi).longValue() != 1)
   {
        e = e.subtract(BigInteger.ONE);
   }
   System.out.println("Clé de chiffrement e = "+e);
   d = e.modInverse(phi);
   System.out.println("Clé de déchiffrement d = "+d);
   for (int i=0;i<chaine_a_crypter.length()-5;i=i+5)
   {
        s = chaine_a_crypter.substring(i,i+5);
        String code = "";
        for (int j=0;j<5;j++)
        {
             char c = s.charAt(j);
             int  n = (int)c;
             if (n<10) code = "0";
             if (n<100) code = code+"0";
             code = code+n;
        }
        BigInteger codage = new BigInteger(code);
        codage = codage.modPow(e,pq);
        chaine_cryptee = chaine_cryptee+codage+";";
   }
   System.out.println("Texte Crypte= "+chaine_cryptee);
   s = "";
   String decodage = "";
   for (int i=0;i<chaine_cryptee.length();i++)
   {
        char c = chaine_cryptee.charAt(i);
        if (c==';')
```

```
        {
             BigInteger codage = new BigInteger(s);
             codage = codage.modPow(d,pq);
             s = codage.toString();
             if (s.length()<15) s="0"+s;
             for (int j=0;j<13;j+=3)
             {
                  String s1 = s.substring(j,j+3);
                  int k = Integer.parseInt(s1);
                  char car = (char)k;
                  decodage = decodage + car;
             }
             s="";
        }
        else s=s+c;
   }
   System.out.println("Texte decrypte : "+decodage);
}
}
```

5.4.4.) L'ALGORITHME DE RABIN-MILLER

L'algorithme de Rabin-Miller est utilisé pour calculer des nombres premiers élevés qui servent à calculer les clefs utilisées par RSA. C'est un algorithme probabiliste qui fonctionne selon le principe suivant:

On choisit un nombre aléatoire p à tester. On calcule b où b est le nombre de fois que 2 divise p-1 (2^b est la plus grande puissance de 2 qui divise p-1). Ensuite, on calcule m tel que $p=1+2^bm$. Puis on exécute l'algorithme suivant:

```
1. On choisit un nombre aléatoire a inférieur à p.
2. On pose j=0 et z=a^m mod p.
3. Si z=1 ou si z=p-1, alors p passe le test est peut-être
   premier.
4. Si j>0 et z=1, alors p n'est pas premier.
5. On pose j=j+1. Si j<b et z #p-1, on pose z=z² mod p et
   retournez à l'étape (4).
6. Si j=b et z#p-1, alors p n'est pas premier.
```

On effectue ce test n fois et si p passe le test n fois, la probabilité pour que p ne soit pas premier est de $1/4^n$.

En java, on teste la primalité d'un nombre à l'aide de la `méthode isProbablePrime(int certainty)`. Le paramètre `Certainty` donne une mesure de la tolérance que l'utilisateur de la méthode peut tolérer. Si la méthode renvoie vrai, alors la probabilité que BigInteger est premier dépasse $(1 - 1/2^{certainty})$. Bien sûr, la vitesse d'exécution de cette méthode dépend de ce paramètre.

On peut construire un nombre premier aléatoire à l'aide du constructeur `BigInteger(int bitlength, int certainty, Random rnd)`.

5.4.5.) L'ALGORITHME D'EUCLIDE ETENDU.

Cet algorithme permet l'obtention d'un inverse modulo n.

Le problème général consiste à trouver un x tel que : **1 = (a*x) mod n,** ce qui s'écrit aussi : **a^{-1} (x (mod n).**
Le calcul de l'inverse modulo n d'un nombre est un problème difficile à résoudre. De plus, il n'existe pas toujours une solution (par exemple : 2 n'a pas d'inverse modulo 14).
En général, **a^{-1} (x (mod n) a une solution x unique si a et n sont premiers entre eux.** Si a et n ne sont pas premiers entre eux, alors **a^{-1} (x (mod n)** n'a pas de solution.
L'algorithme d'Euclide étendu permet le calcul de l'inverse de a modulo n :

```
EUCLIDE_ETENDU(a,n)
SI b=0
  ALORS RETOURNER (a,1,0)
SINON
  (d',x',y') <- EUCLIDE_ETENDU(b,a mod b)
  (d,x,y) <- (d',y',x' - [a/b]y')
  RETOURNER (d,x,y)
```

Signalons enfin que l'algorithme est itératif et peut être très lent pour des grands nombres (complexité $\log_2 n$).

En java, on utilisera la méthode `modInverse(BigInteger m)` qui calcule $\texttt{this}^{-1} \bmod \texttt{m}$.

5.4.6.) BILAN

Ce est positif :

- Un des gros avantages du RSA est d'être un algorithme de chiffrement à clef publique. Il échappe ainsi aux inconvénients majeurs décodage à clef secrète, notamment la transmission des clefs.
- De part sa sécurité, RSA est principalement utilisé pour transmettre les clefs des algorithmes à clef secrète comme le DES ou pour les signatures, ce qui atténue le problème de la transmission des clefs pour de tels algorithmes.
- RSA est actuellement considéré comme incassable en un temps ou avec des moyens raisonnables.

Ce qui est négatif :

- Un premier problème réside dans le choix des deux nombres premiers p et q. Il est en effet préférable de ne pas choisir ces nombres sans précautions. De fait, les concepteurs du système RSA ont donné un certain nombre de règles qu'il est conseillé de suivre lorsqu'il s'agit de choisir le quadruplet (p, q, d, e), car un mauvais choix de ces paramètres peut rendre le système de codage relativement vulnérable et cassable par un bon algorithme de factorisation spécialisé.
  - prendre p et q de tailles sensiblement différentes, mais pas trop.
  - choisir des nombres premiers p et q "sûrs", de la forme 2x + 1, avec x premier, et tel que x - 1 possède de grands facteurs premiers.
  - - choisir d en premier, tel que d et (p -1)*(q - 1) soient premiers entre eux.
  - - choisir ensuite e et vérifier que e >> log2 (n).
  - - ne retenir e que si n et e sont premiers entre eux.
- Le second problème est dû au fait que l'ensemble des nombres premiers connus est fini. Le calcul de nouveaux nombres nécessitant des factorisations dont la complexité est exponentielle, il en résulte que la découverte rapide de nouveaux entiers premiers est informatiquement difficile. Cela met en danger le RSA, dans la mesure où les nombres premiers utilisables ne sont pas si nombreux, et ne doivent pas être utilisés trop souvent, ce qui risquerait de favoriser la cryptanalyse.

Comme tout algorithme de cryptage, le système d'encryption RSA n'élimine pas toutes les formes de régularité détectable au sein de l'information codée. Mais il rend le fichier si obscur à tout système calculatoire polynomial, qu'on estime qu'il est sûr. On n'a pas encore réussi à forcer RSA, mais il

importe d'être vigilant. Les grands organismes de recherche en cryptologie dépensent beaucoup de temps et d'argent pour vérifier l'invulnérabilité des algorithmes à clefs publiques, et donner des règles pour bien les utiliser. Mais leur préoccupation s'oriente surtout autour du point central qui assure la sécurité de RSA : à savoir qu'on ne connaît pas d'algorithme rapide pour calculer la factorisation d'un grand nombre. Même s'il n'est pas prouvé que RSA soit aussi difficile que le problème de la factorisation, si jamais quelqu'un mettait au point un tel algorithme, le RSA serait définitivement abandonné.

5.5.) S.E.T.

5.5.1.) COMMERCE ELECTRONIQUE

La notion de commerce électronique est assez vaste et il est d'autant plus difficile d'en donner une définition exhaustive que les innovations en la matière sont quotidiennes. La seule caractéristique commune tient au fait que le traitement de la commande et du paiement est de nature électronique de bout en bout. Le système s'applique aussi bien à la vente de marchandises que de services. Dans la suite, nous ne faisons pas de différence concernant la nature de l'achat et nous employons le terme «marchandise » pour décrire un bien physique ou une prestation. SET ne se préoccupant pas de la nature exacte de l'échange qui génère la transaction, la distinction ne se justifie en effet pas.

La manière dont l'acheteur sélectionne les marchandises qu'il souhaite acquérir peut varier considérablement. La navigation sur un site Internet n'est qu'une des solutions possibles. L'utilisation d'un CDROM ou d'un traditionnel catalogue papier peut également servir lors de ce choix. Une fois les produits sélectionnés, l'acheteur se voit proposer un bon de commande sur lequel figure le coût total, taxes et frais de port compris. L'acheteur sélectionne ensuite un moyen de paiement accepté par le commerçant et fourni les éléments d'identification le concernant. Nous reviendrons par la suite sur quelques-uns des systèmes existants aujourd'hui avant de nous concentrer sur SET qui constitue sans doute l'avenir. Une fois le formulaire complété, celui-ci est acheminé de manière électronique vers le commerçant qui s'assure de la validité des informations fournies et traite la commande. La communication entre l'acheteur et le commerçant peut se faire par courrier électronique, par utilisation d'une application utilisant un protocole spécifique, ou encore au travers d'un navigateur Web.
Notons dès maintenant que toute transaction met en présence un vendeur et un acheteur. Le premier cherche un moyen simple et peu onéreux d'obtenir le paiement de la marchandise qu'on lui commande. Pour sa part, l'acheteur souhaite avant tout se prémunir des risques liés au transit sur un réseau mondial des informations financières qui le concernent.
Pour être pratique, le système doit permettre un règlement immédiat sous peine de ne pas se différencier de la vente par correspondance traditionnelle.

5.5.2.) LES SYSTEMES ACTUELS.

Le porte-monnaie électronique est un dispositif permettant d'utiliser de l'argent « numérisé ». La mise au point de ce type de dispositif est d'autant plus délicate qu'il convient d'éviter les fraudes. Le porte-monnaie électronique est jugé aujourd'hui comme un instrument indispensable dans le cadre du paiement des petites transactions et/ou des achats anonymes. Les porte-monnaie électroniques, dont le plus utilisé en France est sans doute la carte téléphonique de France Telecom, s'appuient tous sur la notion de prépaiement. Concrètement, le futur acheteur dépose par avance auprès d'un organisme une somme qu'il pourra par la suite utiliser pour effectuer des paiements auprès des commerçants qui ont un accord avec l'organisme en question. Cette approche souffre hélas de plusieurs inconvénients :

- L'acheteur doit fournir par avance la somme qui alimente son portefeuille. Tant que le montant investi se limite à quelques centaines de francs, cela ne pose pas un réel problème. Au delà...

- Lors d'un achat, la somme contenue dans le porte-monnaie électronique de l'acheteur doit être suffisante pour régler le prix de la transaction. A l'heure actuelle, les autorisations de crédit n'existent pas dans ce type de système.
- Les porte-monnaie électroniques sont nombreux et loin d'être universels. Certains ont cependant une véritable notoriété parmi lesquels les systèmes de Globe Online et de DigiCash.

5.5.3.) LE SYSTEMES SET.

Le système SET (Secure Electronic Transaction) constitue une initiative des deux géants des cartes de paiement : MasterCard et Visa. Ces deux groupes sont à l'origine de la spécification de SET, laquelle est aujourd'hui publique. Tout éditeur de logiciel peut donc, en théorie, se lancer dans la réalisation de tout ou d'une partie du système ainsi défini.
L'objectif est de proposer un système de paiement mondial pour le commerce électronique sur la base des cartes de paiement émises par des organismes financiers. Pour qu'un tel projet aboutisse, l'acceptation doit en être la plus large possible, aussi bien en ce qui concerne les commerçants que les acheteurs. De plus, l'utilisation doit en rester simple et le niveau de sécurité être maximal au vue de l'état de l'art en la matière.
En ce qui concerne la sécurité, les problèmes à résoudre sont nombreux. On peut en retenir les trois plus importants :

- La transmission sous forme confidentielle des données financières liées au paiement.
- L'authentification des parties prenantes à la transaction et en particulier, la possibilité mutuelle pour le commerçant et pour l'acheteur de s'assurer de l'identité de l'autre.
- Une garantie d'intégrité des instructions de paiement dès lors que le commerçant et l'acheteur se sont mis d'accord à leur sujet.

La confidentialité repose sur l'utilisation de techniques modernes de cryptographie. L'authentification fait appel à la notion de certificats qui sont au commerce électronique ce que sont les pièces d'identité de la vie courante. La garantie d'intégrité repose sur l'intervention d'une tierce personne qui assure le déroulement financier de la transaction.

5.5.4.) LA TRANSACTION S.E.T.

Dans le système SET, les intervenants sont au nombre de 5.On a :

- **Le titulaire** de la carte de paiement (l'acheteur).
- **Le commerçant** qui propose la marchandise ou le service.
- **L'émetteur** représenté par l'institution financière qui a ouvert un compte à l'acheteur et lui a délivré la carte de paiement.
- **L'acquéreur** représenté par l'institution financière qui a ouvert un compte au commerçant.
- **La gateway** de paiement constitue un matériel destiné à traiter les messages électroniques de paiement en provenance du marchand.

SET entre en jeu à partir du moment où l'acheteur se voit proposer un bon de commande pour les marchandises qu'il souhaite acquérir. Ce bon de commande doit faire apparaître la totalité de la somme due (y compris les taxes et frais de transports).

5.5.5.) LES CERTIFICATS.

Dans le cadre du commerce électronique, chaque intervenant souhaite s'assurer de l'identité de l'autre. Le commerçant veut avoir la certitude que la carte de paiement qui lui est présentée existe effectivement et que l'organisme émetteur en a approuvé l'utilisation dans le cadre du commerce électronique. De même, l'acheteur veut s'assurer que le commerçant est effectivement affilié à un organisme qui lui permet d'accepter ce moyen de paiement. Les deux problèmes sont résolus au moyen de certificats électroniques. Tout intervenant dans le système SET doit obtenir un certificat avant de pouvoir participer à une transaction SET.
Un certificat est constitué d'un ensemble de renseignements concernant la personne dont il prouve l'identité. Ce sont principalement l'identité de la personne et sa clef publique.
Il reste à s'assurer que le certificat est effectivement émis par l'autorité dont il est supposé émaner. Pour ce faire, l'autorité joint son propre certificat à chacun des certificats qu'elle émet. Le certificat de l'autorité provient d'une institution de plus haut niveau, dont on doit pouvoir s'assurer de l'identité... Le système serait sans fin s'il n'existait une super autorité dont le certificat n'est remis en doute par personne.
Les certificats délivrés aux acheteurs constituent une représentation de leur carte de paiement.
Tout intervenant recevant un certificat est tenu d'en vérifier la validité en contactant l'autorité qui l'a émis. Cette dernière valide le certificat et fournit à son tour son propre certificat issu d'une autorité située plus haut dans la hiérarchie, lequel fait lui même l'objet d'une vérification et ainsi de suite jusqu'au sommet de la pyramide. Cette vérification est implicite dans chacun des traitements qui sont détaillés par la suite.

5.5.6.) STRUCTURE D'UN MESSAGE

Nous possédons désormais tous les éléments pour procéder à une communication sécurisée entre deux intervenants. Pour en juger, nous prendrons le cas où Alice souhaite envoyer à Bob un message M. Les opérations se déroulent ainsi :

- Alice calcule le résumé de M. au moyen de la fonction à sens unique.
- Alice produit la signature du message en cryptant le résumé au moyen de l'algorithme à clef publique et en utilisant sa propre clef privée.
- Alice génère de manière aléatoire une clef de codage. Elle utilise cette clef pour crypter au moyen de l'algorithme de cryptage symétrique le message M, la signature et son propre certificat.
- Alice, ayant obtenu le certificat de Bob, connaît la clef publique de ce dernier. Elle l'utilise dans le cadre de l'algorithme à clef publique pour crypter la clef de codage qu'elle a généré précédemment.
- Alice accole le résultat des deux étapes précédentes et envoie le tout à Bob.

Lorsque Bob reçoit le message, il procède de la manière suivante afin d'en récupérer le contenu :

- Bob commence par utiliser sa clef privée et l'algorithme à clef publique pour décoder la clef qui a été générée par Alice.
- Bob utilise l'algorithme symétrique pour décrypter le message M, la signature et le certificat d'Alice.
- Bob récupère le certificat d'Alice et utilise la clef publique qui y figure pour décrypter la signature et obtenir le résumé du message M.
- Bob calcule le résumé de M. au moyen de la fonction à sens unique.
- Il compare enfin la signature qu'il a calculée à celle qu'il a reçue d'Alice afin de vérifier qu'elles sont identiques.

Si l'on examine les différentes étapes, il apparaît que :

- Une personne interceptant l'échange entre Alice et Bob ne peut rien en faire, et en particulier ne peut pas connaître l'identité d'Alice et de Bob. Seul le possesseur de la clef

privée de Bob (c'est à dire Bob lui-même) peut retrouver la clef symétrique qui a été générée aléatoirement.
- Bob peut être certain que la personne à l'origine de l'émission est celle qui possède la clef privée d'Alice (c'est à dire Alice elle même). Si ce n'est pas cette clef qui a été utilisée pour coder la signature, cette dernière une fois décryptée donnera un résumé qui ne sera pas identique à celui que Bob calculera de son côté.
- Si le message M. a été altéré durant son transport, Bob s'en apercevra car le résumé qu'il calculera ne sera pas le même que celui qu'Alice lui aura envoyé.

Chaque intervenant possède en réalité deux jeux de clef publique/privée. L'un est utilisé dans le cadre du calcul de la signature, l'autre est utilisée pour le cryptage des clefs aléatoires générées dans le cadre de l'utilisation de l'algorithme de cryptage symétrique. Cette Petite particularité ne change cependant rien à la cinématique générale. Elle a cependant pour effet de nécessiter deux certificats distincts pour chaque intervenant.

5.5.7.) SIGNATURE DUALE.

Le concept de signature duale permet de résoudre le cas de figure où un intervenant souhaite lier entre eux deux messages adressés à deux intervenants distincts, de sorte que :

- Chacun des deux récepteurs dispose du message~ qui le concerne.
- Chacun des deux récepteurs peut vérifier qu'un résumé que l'autre lui présenterait correspond effectivement à celui de l'autre message.
- Aucun des récepteurs ne peut connaître la signature de l'autre message avant qu'elle ne lui soit transmise par l'autre récepteur.

Pour ce faire, l'émetteur commence par calculer le résumé de chacun des deux messages. L'émetteur concatène ensuite les deux résumés et en calcule la signature. Chacun des deux messages est ensuite acheminé, accompagné de la signature duale le tout étant crypté.

5.5.8.) LE PIPE-LINE.

Au sein du système SET, l'acheteur ne communique jamais directement avec la gateway. Toutes les communications émises par l'acheteur passent en effet par l'intermédiaire du système du commerçant. Nous avons cependant vu que l'acheteur souhaite ne pas divulguer ses informations financières au commerçant. Cette gageure est atteinte grâce à la technique du pipe-line.
SET entre en jeu dès lors que l'acheteur dispose d'une proposition détaillée sur laquelle figure le montant total de la commande. Après en avoir vérifié le contenu et avoir choisi la carte de paiement, l'acheteur requiert du commerçant une copie du certificat de la gateway qui sera en charge de la résolution du paiement. Le commerçant fournit en réponse un numéro de transaction, son propre certificat ainsi que celui de la gateway en les plaçant dans un message qui est crypté selon la technique détaillée au paragraphe précédent.

L'acheteur prépare alors d'une part des informations de commande (qui ne contiennent ni le détail des articles ni les conditions du contrat) et d'autre part son ordre de paiement. Chacun de ces deux messages contient le numéro de transaction qui a été assigné par le commerçant et font tout d'abord l'objet d'une signature duale. Les instructions de paiement (qui contiennent en outre le résumé des informations de commande, au sens cryptographique du terme) sont ensuite cryptées au moyen des informations issues du certificat de la gateway comme si elles faisaient l'objet d'un message à part entière directement adressé à cette dernière. Le résultat de ce cryptage, ainsi que les informations de commande sont rassemblés au sein d'un même message qui est lui même crypté grâce aux informations tirées du certificat du commerçant. Le tout est ensuite expédié à ce dernier.
Lorsque le commerçant reçoit le message, et après l'avoir décrypté, il se retrouve avec des informations de commande « en clair » et des informations de paiement qui restent inintelligibles pour

lui. Il lui faut alors contacter la gateway à laquelle il fournit l'identifiant de la transaction concernée, le montant réclamé, divers renseignements complémentaires, le résumé des informations de commande, ainsi que les informations de paiement encryptées, telles qu'il les a reçues de la part de l'acheteur. Le tout forme un message qui est envoyé à la gateway.
Celle-ci décrypte le message et récupère ainsi « en clair » d'une part les informations fournies par le commerçant et d'autre part les informations de paiement fournies par l'acheteur et qui ont transité par le commerçant. Elle peut vérifier que le résumé des informations de commande tel qu'il est fourni par le commerçant est bien celui qui a servi à générer la signature duale figurant dans les informations de paiement.
Comme nous pouvons le constater, cette approche a permis de résoudre les problèmes suivants :

- Le commerçant n'a pas pu consulter les informations de paiement en provenance du client.
- La gateway n'a connaissance que du montant et non pas des termes de la transaction.
- La gateway a pu vérifier que les informations de paiement qui lui étaient adressées indirectement par le client étaient bien destinées, dans l'esprit de ce dernier, à résoudre la transaction dont le résumé des informations est présenté par le commerçant.
- Si, par la suite, un litige apparaît sur les termes de la commande, le résumé tel qu'il a été fourni par le commerçant doit permettre de lever le doute. il suffit au commerçant ou à l'acheteur de présenter les informations de commande qu'il prétend exactes et de vérifier que le résumé correspond bien à celui qui a été accepté par la gateway au travers de la signature duale.

5.5.9.) SCENARIO TYPE.

L'acheteur fait tout d'abord parvenir sa commande sous la forme d'un message qui contient simultanément la commande ainsi que les éléments nécessaires au paiement. Ces derniers utilisent la technique du pipe-line qui les rend indéchiffrables par le commerçant. Le commerçant extrait les informations de paiement et les transmet à la gateway. Celle-ci décrypte les informations de paiement et les adresse par le canal qu'elle souhaite à l'émetteur de la carte, lequel est alors prié de dire s'il autorise ou non l'utilisation de cette carte pour le montant demandé. Cette transmission entre la gateway et l'émetteur de la carte se fait en dehors du système SET et utilise généralement les réseaux bancaires sécurisés déjà existants. Une fois l'autorisation obtenue, celle-ci est retransmise au commerçant par la gateway. Dès lors, le commerçant possède de la part de la gateway un engagement de paiement de la somme demandée. Cet engagement est matérialisé par un "ticket" informatique. Le paiement lui-même fait l'objet d'une seconde intervention de la part du commerçant qui présente à la gateway le ticket que celle-ci lui a fourni lors de la phase d'autorisation. Cette opération est connue sous le terme de "capture". Bien que l'autorisation et la capture constituent deux opérations décorrélées il est possible de les fusionner au sein d'une seule requête adressée par le commerçant à la gateway. Cette approche se justifie dès lors que le commerçant est certain qu'il peut fournir immédiatement la marchandise requise, qu'il s'agisse d'un service en ligné ou d'un bien qu'i1 possède en stock.
Que la capture soit effectuée immédiatement ou en différé, il est toujours possible pour le commerçant d'émettre un crédit pour le client. Ce cas de figure intervient par exemple lorsque le client renvoi une marchandise qui a été endommagée durant le transport.

5.5.10.) AUTRES SCENARIOS.

D'autres scénarios plus compliqués existent qui prennent en compte des besoins plus pointus.

Le commerçant dispose de la possibilité de réviser le montant de l'autorisation d'une transaction qu'il n'a pas encore capturée. Ce besoin apparaît notamment lorsque les frais de transport ne sont pas connus lors de la commande. Dans ce cas de figure, le commerçant émet une première demande d'autorisation portant sur le montant de la marchandise elle-même sans demander de capture

immédiate. Plus tard, il effectuera une demande de modification du montant autorisé pour prendre en compte les frais de port. Le système SET prend également en compte les paiements effectués en plusieurs versements. Pour qu'un tel mode de paiement soit accepté, les informations de paiement transmises par le client à la gateway doivent cependant mentionner explicitement l'accord du client pour ce faire. En effet, nous avons vu que la technique du pipe-line, utilisée entre la gateway et l'acheteur, garantie que les informations de paiement ne peuvent pas être réutilisées plus tard par le commerçant.
Des reprises manuelles sont enfin possibles afin de traiter les demandes (remboursement émises par les clients après que les éléments concernant transaction ont été purgés du système du commerçant.

6.) Exercices.

Exercice 4.1 :

L'algorithme du Soundex traduit un nom en code 'phonétique'. C'est ainsi que les mots 'mer', mère', 'maire', 'maisre', ... auront le même code. Cet algorithme est d'ailleurs utilisé en généalogie pour retrouver les noms qui auraient subi une transformation due entre autres à une faute de recopie.

L'algorithme du Soundex a été développé au début du siècle par **Margaret K. Odell** et **Robert C. Russel** au bureau américain des archives. Voici les idées sur lesquelles s'appuie l'algorithme:

- les voyelles contribuent moins pour la consonance d'un mot que les consonnes. Elles seront donc supprimées sauf celle en position initiale;
- les lettres H, W ont aussi une contribution minimale et seront donc supprimées sauf celle en position initiale;
- les consonnes redoublées comme NN, SS et MM ou les lettres qui ont la même prononciation peuvent être réduites à une seule occurrence;

Pour savoir si des lettres ont la même consonance, on s'appuie sur la table suivante:

Pour l'anglais		**Pour le français.**	
1	B, F, P, V	**1**	B,P
2	C, G, J, K, Q, S, X, Z	**2**	C,K,Q
3	D, T	**3**	D,T
4	L	**4**	L
5	M,N	**5**	M,N
6	R	**6**	R
		7	G,J
		8	X,Z,S
		9	F,V

Voici un résumé des différentes étapes de l'algorithme:

1. supprimer les éventuels 'espace' initiaux
2. mettre le mot en majuscule
3. garder la première lettre
4. supprimer les lettres A, E, I, O, U, Y, H et W
5. remplacer les lettres restantes par le chiffre associé dans la table
6. supprimer les chiffres répétés (garder une occurence)
7. si le code obtenu contient moins de 4 éléments, compléter à droite par des zéro
8. si le code obtenu contient plus de 4 éléments, conserver les 4 éléments les plus à gauche

Voici quelques exemples de codes obtenus :

Nom	**Soundex pour l'anglais**	**Soundex pour le français**
CAMOS	C520	C580
OINEMORETIME	O563	O563
CROCHET	C623	C623
LAMARTINE	L563	L563

Ecrire un programme calculant le "Soundex" pour un nom donné (passé en paramètre).

Exercice 4.2 :

Il s'agit de calculer pi en utilisant une série attribuée à Euler.

$$p = 2*S(n!)/(1*3*...*(2n+1)), \text{ pour n allant de 0 à l'infini.}$$

Pour ce faire, traduisez en Java le programme basic suivant :

```
DEFLNG a-g : DIM f(8401) AS LONG
a = 10000 : c = 8400
WHILE (b<>c)
  f(b) = a \ 5 : b = b + 1
WEND
WHILE (c>0)
  g = 2*c : d = 0 : b = c
  WHILE (b>0)
    d = d + f(b)*a : g = g - 1 : f(b) = d MOD g
    d = d \ g : g = g - 1 : b = b - 1
    IF (b<>0) THEN d = d*b
  WEND
  c = c - 14 : x$ = STR$ (e + d \ a) : L = LEN (x$)
  PRINT LEFT$ ("0000",5 - L); RIGHTS$ (x$,L - 1);
  e = d MOD a
WEND
```