1.) L'instruction *switch*

Cette instruction permet de choisir une séries d'instructions à exécuter parmi différentes séries. La sélection s'effectue grâce à la valeur d'une variable. Cette variable doit au préalable avoir été déclaré d'un type énumérable tels que byte, short, int ou long. Notez, toute fois, qu'une valeur de type boolean, bien que répondant au critère demandé, n'est pas autorisée. Le choix de la sélection à exécuter se fait en associant des valeurs aux différentes sections de code potentiellement exécutables. Dès lors que l'instruction sera exécutée, la variable considérée aura alors une certaine valeur, et la sélection pourra s'opérer.

Au niveau de la syntaxe, on introduit une telle instruction de la manière suivante : tout d'abord, le mot clé **switch** doit figurer, immédiatement suivit de la variable (d'un type entier) placée entre parenthèses. Ensuite, on doit définir les sélections qui doivent être placées entre accolades. Chacune des sélections se définie ainsi : on place le mot clé **case** suivit d'une valeur (comprise dans les bornes définies par le type utilisé), du caractère : puis d'une série d'instructions à exécuter. On remarquera que la sélection accepte un nombre quelconque d'instructions, par opposition aux autres instructions déjà étudiées.

Une remarque importante est à signaler : il est conseillé que l'instruction break apparaisse en fin de définition de la sélection. Ce n'est en aucun cas une obligation, mais si ce n'est pas le cas, et que la sélection considérée est traitée, le code de la suivante le sera lui aussi, et ce jusqu'à qu'un soit rencontré. Cette propriété peut être utilisée, pourvu que vous en soyez bien conscient. **Si une sélection du switch est lancée son exécution ne s'arretera qu'au prochain break, ou, si l'on en rencontre pas, à la fin de l'instruction switch.**

```
public class Switch {
    public static void main(String argv[]){
        for(int i=0;i<10;i++)
        switch(i){
           case 9: System.out.println("Neuf"); break;
           case 8: System.out.print("Huit"); System.out.println(""); break;
           case 7: { System.out.print("Sept"); System.out.println(""); } break;
           case 6: System.out.print("Six ");
           case 5: System.out.println("Cinq"); break;
           case 4:
           case 3: System.out.print("Quatre Trois ");
           case 2: System.out.println("Deux"); break;
           case 1: System.out.println("Un"); break;
           default: System.out.println("Zero");
        }
    }
}
```

2.) Visibilités des variables.

Tout objet, variable ou fonction doit être défini et déclaré avant son utilisation. Mais ces objets ont des statuts divers et il faut soigneusement considérer pour chacun d'eux les caractéristiques suivantes :

- Sa **durée de vie** qui peut être soit permanente, c'est à dire que l'objet existe durant toute la durée de l'éxécution du programme, soit temporaire c'est à dire que l'objet n'existe que temporairement pendant la durée d'éxécution d'un bloc.
- Sa **visibilité**, appelée également sa portée ou son espace de validité, c'est à dire les parties du programme où cet objet est utilisable, c'est à dire connu et référençable.

D'une façon générale, la variable est visible à l'intérieur du bloc où elle est définie. Un bloc est définit comme l'ensemble des instructions comprises entre deux accolades { } .

Si dans un bloc on redéfinit une variable existant dans un bloc supérieur, cette nouvelle variable masque la variable supérieur à l'intérieur de ce bloc (donc aussi pour ses sous-blocs).

3.) Les tableaux.

Comme les vecteurs de Caml, les tableaux sont des collections d'objets de même type. Un fois le tableau créé, il n'est pas possible (du moins facilement) de changer sa taille.

Un tableau est créé à l'aide de l'opérateur new. Par exemple, le code suivant créé un tableau de 100 entiers :

```
int [] unTableau = new int[100];
```

Java offre un raccourci pour créer un objet tableau et d'initialiser simultanément ses éléments. Voici la syntaxe :

```
String[] lesPoetes = {"Deschamps","D'Orléans","Villon","Marot","Scève","Du
Bellay","Ronsard","Labé","Garnier","Desportes","Garnier","D'Aubigné"};
```

On accède de la manière suivante aux éléments du tableau :

```
String mon_préféré = lesPoetes[2];
```

(Le premier élément du tableau a l'indice 0).

O peut déclarer des tableaux multidimensionnels :

```
String[][] lesPoetes = {{"Deschamps","1346-1406"}, {"D'Orléans","1394-1465"}};
```

ou

```
int[][] unTableau = new int[10][10];
```

4.) Les chaines de caractères.

Les chaînes sont des suites de caractères délimitées par des guillemets. Exemple de chaîne :

```
"En cette foi je veux vivre et mourir"
```

Java ne propose pas de type chaîne mais fourni une classe prédéfinie appelée String. Chaque chaîne entre guillements est une instance de la classe String. On peut donc lui appliquer des méthodes :

```
System.out.println("En cette foi je veux vivre et mourir".toLowerCase();
```

Java autorise plusieurs raccourcis pour la manipulation de chaînes. Par exemple, une nouvelle chaîne peut être créé des deux manières suivantes :

```
String chaine1 = new String("En cette foi je veux vivre et mourir");
String chaine2 = "En cette foi je veux vivre et mourir";
```

De même, Java autorise l'emploi du signe + pou concaténer des chaînes :

```
String chaine 3 = "En cette foi "+"je veux vivre"+ " et mourir";
```

Lorsqu'on concaténe une chaîne et une valeur qui n'est pas une chaîne, cette valeur est convertie en chaîne. Voici un exemple qui concaténe une chaine, un nombre, une chaine, un nombre et un caractère

```
chaine3 = "François Villon "+1431+" mort aprés "+1463+'.';
```

Pour extraire une sous-chaîne d'une chaîne, on utilise la méthode substring :

```
chaine3 = chaine3.substring(10,6);
```

Comme en C et en Caml, le premier caractère d'une chaîne est en position 0.

Pour connaître la longueur d'une chaine, on utilise la méthode length :

```
int longueur = chaine3.length();
```

5.) Etude de la classe BigInteger.

La classe BigInteger se trouve dans le package java.math.

Elle hérite de la classe java.lang.Number

Il s'agit d'une classe publique.

Ce n'est pas une classe finale. Elle est donc héritable.

La classe BigInteger permet de représenter des nombres entiers avec une précision arbitraire.

Champs	
`static BigInteger`	ONE Constante représentant le chiffre 1.
`static BigInteger`	ZERO Constante représentant le chiffre 0.

Constructeurs	
BigInteger(`int bitLength, int certainty,` Random `rnd`) Création d'un objet BigInteger aléatoire, positif et probablement premier.	
BigInteger(`int numBits,` Random `rnd`) Création d'un objet BigInteger aléatoire, positif et compris dans l'intervalle allant de `0` à $(2^{numBits} - 1)$, inclus.	
BigInteger(String `val`) Création d'un objet BigInteger à partir d'une chaîne de caractères.	
BigInteger(String `val, int radix`) Création d'un objet BigInteger à partir d'une chaîne de caractères dans une base radix	

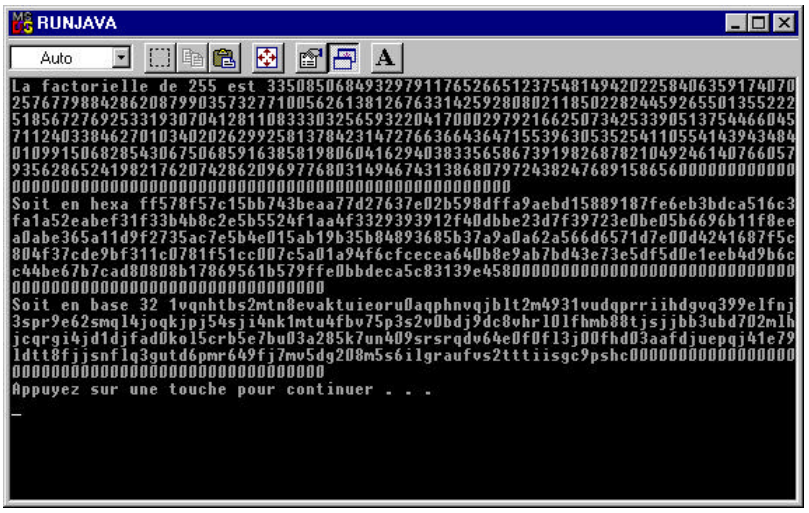
Methodes	
`BigInteger`	abs() Valeur absolue.
`BigInteger`	add(BigInteger `val`) Addition.
`BigInteger`	and(BigInteger `val`) Et binaire.
`int`	bitCount() Nombre de bits dans une représentation en complément à 2
`int`	compareTo(BigInteger `val`) Comparaisons. Renvoie -1, 0 or 1 selon que cet objet BigInteger est plus petit, égal ou plus grand que `val`.
`int`	compareTo(Object `o`) Comparaison avec un objet.
`BigInteger`	divide(BigInteger `val`) Division entière
`BigInteger[]`	divideAndRemainder(BigInteger `val`) Renvoie le quotient entier et le reste.
`double`	doubleValue() Conversion en double

`boolean`	equals(Object `x`) Test d'égalité
`float`	floatValue() Conversion en float.
`BigInteger`	gcd(BigInteger `val`) pcgd.
`int`	intValue() Conversion en un entier.
`boolean`	isProbablePrime(`int certainty`) Renvoie true si l'objet BigInteger est probablement un nombre premier.
`long`	longValue() Conversion en un entier long.
`BigInteger`	mod(BigInteger `m`) Reste de la division entière.
`BigInteger`	modPow(BigInteger `exponent`, BigInteger `m`) Renvoie un objet BigInteger de valeur ($this^{exponent} \bmod m$).
`BigInteger`	multiply(BigInteger `val`) Multiplication
`BigInteger`	negate() Négation.
`BigInteger`	or(BigInteger) `val`) Ou binaire.
`BigInteger`	pow(`int exponent`) Puissance.
`BigInteger`	shiftLeft(`int n`) Décalage à gauche.
`BigInteger`	shiftRight(`int n`) Décalage à droite
`int`	signum() Renvoie le signe (-1,0 ou +1).
`BigInteger`	subtract(BigInteger `val`) Soustraction.
`byte[]`	toByteArray() Conversion en tableau de bits.
`String`	toString() Conversion en chaîne de caractères.
`String`	toString(`int radix`) Conversion en chaine de caractères dans une base donnée.
`BigInteger`	xor(BigInteger `val`) Xor

Exemple d'utilisation de la classe BigInteger : Calcul d'une factorielle.

```
import java.math.*;

public class Factorielle
{
  public static void main(String argv[])
  {
    BigInteger factorielle = new BigInteger("1");
    BigInteger compteur = new BigInteger("FF",16); // 255 en hexadécimal
    while (! compteur.equals(BigInteger.ZERO))
    {
      factorielle = factorielle.multiply(compteur);
      compteur = compteur.subtract(BigInteger.ONE);
    }
    System.out.println("La factorielle de 255 est "+factorielle);
    System.out.println("Soit en hexa "+factorielle.toString(16));
    System.out.println("Soit en base 32 "+factorielle.toString(32));
  }
}
```



6.) Etude de la classe BigDecimal.

La classe BigDecimal se trouve dans le package java.math.

Elle hérite de la classe java.lang.Number

Il s'agit d'une classe publique.

Ce n'est pas une classe finale. Elle est donc héritable.

La classe BigDecimal permet de représenter des nombres décimaux signés avec une précision arbitraire. Cette classe fourni des opérations pour l'arithmétique de base.

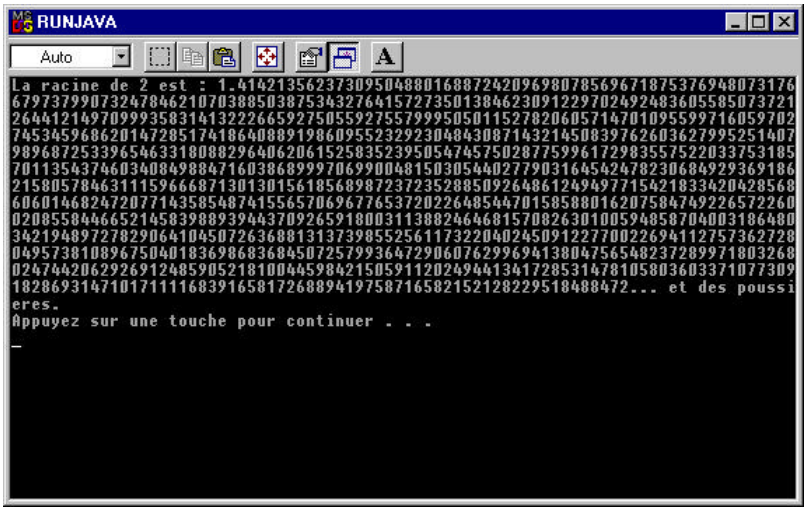
Champs	
`static int`	ROUND_CEILING Arrondir vers +∞ Rounding mode to round towards positive infinity.
`static int`	ROUND_DOWN Arrondir vers 0
`static int`	ROUND_FLOOR Arrondir vers -∞

Constructeurs	
BigDecimal(BigInteger `val`) Construit un objet BigDecimal à partir d'un BigInteger.	
BigDecimal(BigInteger `unscaledVal, int scale`) Construit un objet BigDecimal à partir d'un BigInteger avec une précision scale.	
BigDecimal(`double val`) Construit un objet BigDecimal à partir d'un double.	
BigDecimal(String `val`) Construit un objet BigDecimal à partir d'un objet String.	

Methodes	
`BigDecimal`	abs() Valeur absolue.
`BigDecimal`	add(BigDecimal `val`) Addition.
`int`	compareTo(BigDecimal `val`) Comparaison.
`BigDecimal`	divide(BigDecimal `val, int roundingMode`) Division.
`BigDecimal`	divide(BigDecimal `val, int scale, int roundingMode`) Division en spécifiant la précision.
`double`	doubleValue() Conversion en double.
`boolean`	equals(Object `x`) Test d'égalité.
`float`	floatValue() Conversion en flottant.

`int`	intValue() Converts this BigDecimal to an int.
`long`	longValue() Conversion en entier long.
`BigDecimal`	max(BigDecimal `val`) Maximum
`BigDecimal`	min(BigDecimal `val`) Minimum.
`BigDecimal`	movePointLeft(`int n`) Déplacement de la virgule de n positions à gauche.
`BigDecimal`	movePointRight(`int n`) Déplacement de la virgule de n positions à droite.
`BigDecimal`	multiply(BigDecimal `val`) Multiplication.
`BigDecimal`	negate() Négation.
`int`	scale() Renvoie la précision.
`BigDecimal`	setScale(`int scale`) Fixe la précision.
`BigDecimal`	setScale(`int scale, int roundingMode`) Fixe la précision en utilisant le mode d'arrondi.
`int`	signum() Renvoie le signe
`BigDecimal`	subtract(BigDecimal `val`) Soustraction.
`BigInteger`	toBigInteger() Conversion en objet BigInteger.
`String`	toString() Conversion en objet String..
`BigInteger`	unscaledValue() Conversion en BigInteger.

Exemple d'utilisation : Calcul de la racine de 2.



```
import java.math.*;

public class Racine
{
  public static void main(String argv[])
  {
    BigDecimal nombre = new BigDecimal(2);
    BigDecimal deux = new BigDecimal(2);
    BigDecimal resultat = new BigDecimal(1);
    BigDecimal temp = new BigDecimal(0);
    int precision = 1000;
    resultat.setScale(precision);
    temp.setScale(precision);
    for (int i = 1;i<5000;i++)
    {
          temp = nombre.divide(resultat,precision,BigDecimal.ROUND_DOWN);
          temp = temp.add(resultat);
          resultat = temp.divide(deux,precision,BigDecimal.ROUND_DOWN);
    }
    System.out.println("La racine de 2 est : "+resultat.toString()+"... et des
poussieres.");
  }
}
```

7.) La cryptologie.

7.1.) Position du problème.

Il est parfois sage et indispensable de cacher certaines choses. Dès l'origine le mystère a fasciné l'Homme. Ce qui est dissimulé ou caché l'intéresse énormément, car sa soif de savoir est sans limites. Mais, celui qui sait, comme celui qui dissimule possède une forme de pouvoir.
D'autre part, à tous les niveaux, il semble important d'empêcher le premier venu de lire tel ou tel document. De l'enfant qui ne veut pas que son petit frère lise ses lettres aux nations qui veulent conserver leurs secrets, la volonté de crypter est présente partout. Aussi se sont développés des moyens de rendre illisibles des informations : ce sont les algorithmes de cryptage ou cryptosystèmes.

7.1.1.) VOCABULAIRE.

Comme pour une lettre postale, la personne qui envoie un message éventuellement codé s'appelle l'expéditeur. La personne qui le reçoit s'appelle le destinataire. Le message lisible est appelé texte en clair. Par un processus appelé chiffrement ou cryptage il est rendu illisible : on l'appelle alors cryptogramme ou texte chiffré ou même tout simplement chiffre. Le processus inverse qui à partir du chiffre renvoie le texte en clair initial est appelé déchiffrement ou décryptage.

L'art et la science de garder le secret des messages est appelé cryptographie et est pratiquée par des cryptographes. Les cryptanalystes exercent la cryptanalyse qui est l'art de décrypter des messages chiffrés. La branche des mathématiques qui traite de la cryptographie et de la cryptanalyse s'appelle la cryptologie. Par la force des choses, ses pratiquants, appelés cryptologues, sont maintenant presque tous des mathématiciens théoriciens.

Le Petit Larousse précise que la cryptographie est "l'ensemble des techniques permettant de protéger une communication au moyen d'un code graphique secret".

7.1.2.) POURQUOI CRYPTER ?.

L'enfant cryptera ses lettres par soucis d'intimité mais aussi pour embêter son petit frère. L'intérêt du cryptage dans ce cas, même s'il est louable, est peu important.
En revanche, les banques pour assurer la confidentialité des opérations de leurs clients, les gouvernements pour assurer le secret de leurs communications internes et diplomatiques, les armées pour interdire à l'ennemi de connaître leurs plans, les utilisateurs de tous types de cartes avec code pour interdire aux escrocs de détourner leurs biens, … sont des exemples d'organismes ou de personnes pour lesquels le cryptage est indispensable. De plus, le flux croissant de données transitant dans les réseaux internationaux impose de crypter ces informations pour en assurer l'intégrité et la confidentialité. C'est un des problèmes de l'Internet. Enfin, le cryptage permet, lorsqu'il est utilisé comme signature numérique, d'authentifier l'auteur d'un document, c'est-à-dire d'être sûr qu'un chiffre provient bien d'un expéditeur unique connu.
En bref, le cryptage permet d'éviter l'interception, la lecture ou la modification d'un message en clair, ainsi que la fabrication d'un message factice..

7.2.) Techniques cryptographiques.

Avant l'arrivée des ordinateurs, la cryptographie existait mais restait relativement simple : elle se contentait de substitutions ou de transpositions de caractères, les meilleurs systèmes combinant les deux plusieurs fois. Ces systèmes cryptographiques étaient très vulnérables, mais à cette époque il n'y avait pas encore de cryptanalyse. C'est en fait le développement de la cryptanalyse qui à donné les bases mathématiques nécessaires à la conception d'algorithmes sûrs.

Aujourd'hui, tous ces systèmes sont abandonnés à cause de leur vulnérabilité, mais les méthodes restent les mêmes : des substitutions et des transpositions de caractères d'un alphabet à 2 caractères, l'alphabet binaire. En effet, tout message peut se mettre sous forme de suite de caractères, lesquels sont codés sur 8 bits. Donc tout message est en fait une succession de bits donc est codé avec un alphabet à 2 caractères. Les techniques modernes font donc des manipulations sur ces bits en utilisant les résultats de la théorie des nombres qui était naguère inusité. De plus, les nouveaux systèmes tirent pleinement parti du potentiel de calcul des machines actuelles.
Ce sont désormais des méthodes très sures qui sont utilisées, mises au point par des experts. Il ne faut pas croire qu'il est simple de faire soi-même un algorithme de cryptage sûr quand on est néophyte en cryptanalyse.

7.3.) Les differents types d'algorithmes cryptographiques

Un algorithme cryptographique est une fonction mathématique utilisé pour le chiffrement et pour le déchiffrement. Pour crypter un message, on peut utiliser un algorithme de chiffrement et pour déchiffrer un texte crypté on peut lui appliquer un algorithme de déchiffrement. De manière assez simple, on peut dire qu'il existe trois grands types d'algorithmes cryptographiques.

7.3.1.) LES ALGORITHMES RESTREINTS

Ce sont les algorithmes tenus secrets, c'est-à-dire les systèmes cryptographiques connus d'un petit nombre de personnes souvent juste de l'expéditeur et du destinataire. Ce type de systèmes est sûr tant que l'algorithme n'est pas découvert par l'ennemi. En revanche, il devient inutile dès que l'ennemi le découvre. Or il n'est pas simple de maintenir secret un algorithme. De plus, à l'heure actuelle, ce genre de système est inadéquat car un nombre croissant de communications se font entre des personnes qui se connaissent peu. Cela impose d'avoir autant d'algorithmes différents que de personnes à qui on envoie des messages ce qui n'est pas pratique. En outre, ce type de système est souvent facile à casser pour des cryptanalystes expérimentés, et si un des utilisateurs du système révèle le secret, tout le système s'effondre. Donc ce genre de cryptosystème n'est pas très utilisé car il est vulnérable, mais certains cryptages vidéo comme Zénith sont des exemples d'algorithmes restreints : il n'est pas nécessaire que ces codages soit sûr à 100%.

7.3.2.) LES ALGORITHMES A CLE SECRETE.

Ce sont souvent des algorithmes publics, c'est-à-dire connus de tous, dont la sécurité repose sur l'utilisation d'une clef secrète. Une clef est une information supplémentaire qui participe à l'algorithme de cryptage ou de décryptage. C'est souvent une chaîne caractères. Dans les cryptosystèmes à clef privée, la clef de chiffrement peut être calculée à partir de la clef de déchiffrement et vice versa. Souvent même les 2 clefs sont identiques. Ce type de système engendre 3 difficultés : le vol de la clef par l'ennemi ; la distribution des clefs ; la multiplication des clefs qui doivent rester secrètes quand on communique avec beaucoup d'interlocuteurs. Il faut donc arriver à conserver ces clefs dans un endroit sûr et penser à les changer souvent pour ne pas que l'ennemi puisse avoir le temps de les trouver. Le problème de la transmission des clefs demeure quelque chose de très difficile à résoudre puisque toute la sécurité du système repose sur le secret de la clef.

7.3.3.) LES ALGORITHMES A CLE PUBLIQUE.

Ce sont, là encore, souvent des algorithmes publics mais où les clefs de chiffrement et de déchiffrement sont différentes. Il n'est pas possible de calculer la clef de déchiffrement avec des moyens raisonnables à partir de la clef de chiffrement. Les deux clefs ne sont pas secrètes : l'une, appelée clef publique est connue de tous ; l'autre, appelée clef privée n'est connue que d'une personne. La clef publique dépend du destinataire : il y a une clef publique par destinataire, celle-ci étant stockée dans une sorte de base de données. Tout le monde peut consulter cette base de données mais on ne peut modifier que sa clef publique. La clef privée doit être maintenue secrète. Quand la clef de

chiffrement est la clef publique et la clef de déchiffrement est la clef privée, le système cryptographique ainsi constitué permet à n'importe qui d'envoyer un message crypté à quelqu'un, mais seule cette personne pourra lire le message car c'est la seule qui détient la clef de déchiffrement. Dans l'autre cas – i.e. la clef de déchiffrement est la clef publique et la clef de chiffrement est la clef privée – seule une personne pourra coder un message que tout le monde pourra lire : cela permet d'authentifier l'auteur d'un message, c'est ce que l'on appelle une signature numérique.

7.4.) Les différentes méthodes utilisées dans les algorithmes de cryptage.

Les 2 techniques décrites ci-après ne dépendent pas de la base de l'alphabet choisi : que l'alphabet comporte 26 lettres ou qu'il n'en comporte que 2 comme l'alphabet binaire, les techniques restent les mêmes.

7.4.1.) LES SUBSTITUTIONS.

Une substitution est une opération au cours de laquelle on remplace un caractère par un autre. Par la substitution réciproque, on retrouve le texte de départ. Il existe 4 types de substitutions.

Les substitutions simples

A un caractère du texte en clair on associe un unique caractère du texte chiffré. C'est par exemple ce qui est utilisé dans les cryptogrammes des journaux. C'est le procédé utilisé par l'algorithme « César » décrit plus loin.

Les substitutions homophoniques ou substitutions simples à représentation multiple

A un caractère du texte en clair on associe plusieurs caractères du texte chiffré. Par exemple, à la lettre "A" on fait correspondre 6 ou 15 ou 96, à la lettre "B" 8 ou 56 ou 963…

Les substitutions polyalphabétiques

On utilise plusieurs substitutions simples : celle qui est utilisée dépend de la position du caractère dans le texte en clair. C'est le procédé utilisé dans l'algorithme de Vigenère.

Les substitutions simples par polygrammes

On ne chiffre plus caractère par caractère, mais on considère plusieurs caractères d'un coup. Par exemple, "AAA" est transformé en "TOTO", "LUC" est transformé en "NICOLAS"…

7.4.2.) LES TRANSPOSITIONS

Les caractères du texte en clair sont inchangés mais leurs positions respectives sont modifiées. On peut par exemple prendre le message et l'écrire en colonne au lieu de l'écrire en ligne et on envoie le message ligne par ligne. Prenons le message « DEMAIN DES L AUBE A L HEURE OU BLANCHIT LA CAMPAGNE ». On l'écrit :

```
D N A L E A T M E
E D U H O N L P
M E B E U C A A
A S E U B H C G
I L A R L I A N
```

Soit le chiffre : « DNALEATMEEDUHONLPMEBEUCAAASEUBHCGILARLIAN »

7.5.) Quelques notions de cryptologie.

Pour compléter un peu notre présentation de la cryptographie, nous allons introduire quelques notions supplémentaires en nous bornant à une simple description de vocabulaire. Nous ne souhaitons pas introduire ici de lourds concepts mathématiques appuyés de démonstrations.

7.5.1.) FONCTION A SENS UNIQUE.

Une fonction à sens unique est une fonction f de la variable x qui possède les caractéristiques suivantes :

- Etant donné x il est facile de calculer la valeur f(x) ;
- Etant donné f(x) il est difficile de calculer la valeur de x.

La facilité est un compromis entre temps de calcul et puissance de calcul nécessaire. Les fonctions utilisées pour l'authentification sont des exemples de fonctions à sens unique. Mathématiquement nous n'avons aucune preuve de l'existence d'une telle fonction.

7.5.2.) FONCTION A SENS UNIQUE A BRECHE SECRETE.

Une fonction à sens unique à brèche secrète est une fonction à sens unique f qui possède en plus la caractéristique : étant donné f(x) et une information supplémentaire y on peut calculer x facilement.

Les algorithmes de cryptage à clef publique sont des exemples plus significatifs dans notre propos, la clef privée étant la brèche secrète.

7.5.3.) FONCTION DE HACHAGE

La dénomination n'est pas unique puisqu'on parle de fonction de compression, de fonction de contraction, de digest, d'empreinte digitale, de code correcteur cryptographique, de code de vérification d'intégrité, de code de détection de manipulation et de code d'authentification. C'est une fonction qui à partir d'un élément, souvent une chaîne de caractères, renvoie un autre élément souvent de taille inférieure fixe. Si 2 éléments à peu près semblables renvoie la même empreinte de hachage on peut raisonnablement dire qu'ils sont égaux.

7.5.4.) FONCTION DE HACHAGE A SENS UNIQUE.

C'est une fonction de hachage qui est de plus une fonction à sens unique. Ce sont souvent des fonctions pseudo aléatoires. Ces fonctions sont utilisées pour authentifier un utilisateur. On distingue 2 types de fonctions de hachage à sens unique :

- Fonction de hachage à sens unique avec clef :L'empreinte est fonction à la fois de la chaîne d'entrée et de la clef.
- Fonction de hachage à sens unique sans clef : L'empreinte est uniquement fonction de la chaîne d'entrée.

7.6.) La sécurite.

7.6.1.) QU'EST CE QU'UN BON ALGORITHME DE CRYPTAGE.

Le problème de la sécurité est le point central de tout algorithme de cryptage. En effet, le but du cryptage est d'empêcher à quiconque de pouvoir lire le chiffre. La sécurité d'un algorithme représente, en gros, la difficulté qu'un cryptanalyste aura à décrypter un chiffre.

On dira qu'un algorithme est sûr si le temps et l'argent nécessaire au déchiffrement pirate dépassent la valeur de l'information chiffrée. Un algorithme invulnérable dans la pratique est dit sûr. On dira qu'un algorithme est dit inconditionnellement sûr si le temps de forçage ne dépend pas de la quantité de texte chiffré dont on dispose. Enfin, on dira qu'un algorithme est invulnérable par calcul ou fort s'il ne peut être cassé avec les ressources disponibles actuellement et dans le futur.

On appelle effort le produit temps de calcul par la puissance mise en jeu. Par exemple pour tester 2^{128} possibilités, l'effort est 10^9 fois l'âge de l'univers avec 1 million de tests par seconde.

Selon le degré de sécurité que l'on recherche, on devra utiliser des algorithmes plus ou moins performants. Le choix d'un cryptosystème est donc quelque chose de très important et de prioritaire. Pour conclure, on peut dire qu'une sécurité absolue repose sur 3 briques :

- La qualité du brouillage c'est-à-dire de l'efficacité de l'algorithme ;
- Le nombre de solutions possibles correspondant au nombre de clefs possibles ;
- Le temps de forçage pour tester toutes les solutions ou plus précisément l'effort nécessaire pour forcer le cryptosystème.

7.7.) Principes de securite.

7.7.1.) PROTECTION DES COMPOSANTES DU SYSTEME DE CRYPTAGE

Certains éléments doivent nécessairement rester secrets. Un cryptosystème c'est le choix d'un algorithme et le choix d'une clef plus ou moins grande. La clef est la sécurité première : elle doit donc être maintenue secrète. La sécurité, au deuxième niveau, est celle de la transmission des clefs au destinataire, l'algorithme étant supposé connu de lui. C'est un problème non trivial sauf dans le cas des algorithmes à clefs publiques : dans ce seul cas en effet il n'y a pas de transmission de clef puisque seul le destinataire connaît sa clef privé correspondante à la clef publique utilisée pour crypter.

7.7.2.) LE TEMPS DE FORÇAGE

Un algorithme est sûr si le temps de forçage est très supérieur à l'échelle des temps des activités humaines. Un bon brouillage rend la cryptanalyse impuissante ce qui impose de tester toutes les solutions. Mais il faut faire attention car c'est une tâche parallélisable : si on augmente le nombre de

machines mises en jeu, le temps de forçage diminue d'autant. Cependant, il faut tenir compte du temps mis pour déchiffrer.

7.7.3.) VULNERABILITE D'UN SYSTEME DE CRYPTAGE

La cryptanalyse a deux facettes :

- Attaque à texte chiffré connu : On cherche les régularités, les fréquences des caractères,…
- Attaque à texte chiffré et clair connus : On essaie de trouver des correspondances entre les deux textes.

Cryptanalysement parlant, un codage est bon si :

- On crypte les données par bloc de plusieurs caractères : cela diminue l'efficacité de la cryptanalyse différentielle ;
- Le codage des blocs est instable, c'est à dire si 2 blocs très semblables ont des chiffres très différents ;
- On utilise plusieurs algorithmes de cryptage sur un bloc avec un ordre fourni par une clef de sûreté.
- Le seul moyen de retrouver le texte initial à partir du texte chiffré est d'essayer toutes les clefs.

7.8.) Quelques algorithmes à substitution.

7.8.1.) CESAR.

César est le premier système de cryptage connu. Selon la légende, Jules César l'aurait utilisé au cours de ses combats afin de transmettre des messages codés qui ne soient pas déchiffrables par ses adversaires.

César étant le premier algorithme de cryptage à être apparu, il est tout à fait logique qu'il soit simple.

En effet, c'est un algorithme de cryptage par substitution. Le mode de cryptage consiste à remplacer une lettre de l'alphabet par la lettre qui se situe n places plus loin dans l'alphabet (historiquement, Jules César utilisait un décalage de 3).

Par exemple, si on prend n=3, on aura les substitutions suivantes:

A → D ; B → E ; C → F ; … ; Y → B ; Z → C

Ainsi, le programme reçoit la chaîne de caractères que constitue le message puis ajoute 3 aux valeurs des caractères, que l'on obtient grâce à la table des caractères ASCII (et en se ramenant à une numérotation de 0 à 25 des lettres de l'alphabet), et retranscrit cette nouvelle valeur sous forme d'un caractère.

Nous devons toutefois noter que l'algorithme Cesar ne crypte que les 26 lettres de l'alphabet et remplace toutes les autres par des espaces.

```
/**
 Cryptage par l'algorithme "César"
 @author     LWH
 @version    1.0
*/

public class Cesar {

public static void main(String[] argv) {

    final String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    final String codage   = "DEFGHIJKLMNOPQRSTUVWXYZABC";

    String chaine_a_crypter = " ";
    String chaine_cryptee = new String();
    char c;
    int pos;

    if (argv.length > 0)
         chaine_a_crypter = argv[0]; else chaine_a_crypter = "La reine blanche
comme lys";

    chaine_a_crypter = chaine_a_crypter.toUpperCase();

    for (int i=0;i<chaine_a_crypter.length();i++)
    {
          c = chaine_a_crypter.charAt(i);
          pos = alphabet.indexOf(c);
          if (pos >= 0)
          {
                c = codage.charAt(pos);
                chaine_cryptee = chaine_cryptee + c;
          }
          else
          {
                chaine_cryptee = chaine_cryptee + " ";
          }
    }
    System.out.println("Le texte à crypter est : "+chaine_a_crypter);
    System.out.println("Le texte crypté est    : "+chaine_cryptee);
    }
}
```

Le seul avantage de cette méthode est sa simplicité. En effet, elle ne met en jeu aucun calcul ou permutation compliqué.
C'est aussi ce qui fait sa faiblesse: n'importe qui peut, sans avoir besoin d'une machine, déchiffrer un message codé en essayant tout simplement toute les clefs de 1 à 25. C'est pourquoi cette méthode est à proscrire dès que le veut vraiment rendre confidentiel un message.

7.8.2.) VIGENERE

Vigenere a inventé son code au XVIème siècle selon un procédé voisin de celui de César. En effet, dans le cas du code de César nous sommes confrontés à un code à substitution simple; ici, Vigenere emploie un code à substitution polyalphabétique.

Le code de Vigenere étant un code à substitution polyalphabétique, l'algorithme consiste à substituer à chaque lettre de notre message une lettre de l'alphabet que nous calculons à partir d'une clef.

```
/**
 Cryptage par l'algorithme "Vigenere"
 @author     LWH
 @version    1.0
*/

public class Vigenere {

public static void main(String[] argv) {

    final String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String table[] = new String[26];

    // Remplissage de la table des alphabets
    table[0]  = "AYXWVUTSRQPONMLKJIHGFEDCBZ";
    table[1]  = "ACBDEFGHIJKLMNOPQRSTUVWXYZ";
    table[2]  = "ZYXWVUTSRQPONMLKJIHGFEDCBA";
    table[3]  = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    table[4]  = "ZYXWVUTSRQPONMLKJIHGFEDCBA";
    table[5]  = "CDEFGHIJKLMNOPQRSTUVWXYZAB";
    table[6]  = "OPQRSTUVWXYZABCDEFGHIJKLMN";
    table[7]  = "STUVWXYZABCDEFGHIJKLMNOPQR";
    table[8]  = "AZBYCXDWEVFUGTHSIRJQKPLOMN";
    table[9]  = "BYCXDWEVFUGTHSIRJQKPLOMNAZ";
    table[10] = "NZBYCXDWEVFUGTHSIRJQKPLOMA";
    table[11] = "GTHSIRJQKPLOMNAZBYCXDWEVFU";
    table[12] = "AZBYCXDWEVFUGTHSIRJQKPLOMN";
    table[13] = "NZBYCXDWEVFUGTHSIRJQKPLOMA";
    table[14] = "AZBYCXDWEVFUGTHSIRJQKPLOMN";
    table[15] = "MNOPQRSTUVWXYZABCDEFGHIJKL";
    table[16] = "LMNOPQRABCDEFGHIJKSTUVWXYZ";
    table[17] = "FGHIJKLMNABCDEOPQRSTUVWXYZ";
    table[18] = "VWXYABCDEFGHIJKLMNOPQRSTUZ";
    table[19] = "IJKLMNOPQABCDEFGHRSTUVWXYZ";
    table[20] = "AFGHIJKBCDELMNOPQRSTUVWXYZ";
    table[21] = "KJIHGFZYXWVUTSRQPONMLEDCBA";
    table[22] = "KJIHGFEDCBAZYXWVUTSRQPONML";
    table[23] = "LKJIHGFZYXWVUTSRQPONMEDCBA";
    table[24] = "TSRQPONMLKJZYXWVUIHGFEDCBA";
    table[25] = "ZYXWVUTSRQPONMLKJIHGFEDCBA";

    String chaine_a_crypter = " ";
    String cle = " ";
    String chaine_cryptee = new String();
    char c;
    int pos,j,k;


    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";

    chaine_a_crypter = chaine_a_crypter.toUpperCase();

    j=0;
    for (int i=0;i<chaine_a_crypter.length();i++)
    {
          c = chaine_a_crypter.charAt(i);
          pos = alphabet.indexOf(c);
```

```
            if (pos >= 0)
            {
                  // Choix de l'alphabet k
                  if (j >= cle.length()) j=0;
                  c = cle.charAt(j);
                  k = alphabet.indexOf(c);
                  j++;

                  // Codage de la chaine dans l'alphabet
                  c = table[k].charAt(pos);
                  chaine_cryptee = chaine_cryptee + c;
            }
            else
            {
                  chaine_cryptee = chaine_cryptee + " ";
            }
      }
      System.out.println("Le texte à crypter est : "+chaine_a_crypter);
      System.out.println("Le texte crypté est    : "+chaine_cryptee);
       }

}
```

Tout comme Cesar, le programme Vigenere ne code que les lettres de l'alphabet, c'est pourquoi, j'ai choisi de laisser inchangés les autres caractères.

L'avantage de cet algorithme par rapport à César est qu'il est un peut plus compliqué à déchiffrer du fait de l'utilisation d'une clef secrète. Cependant, il reste néanmoins très facilement déchiffrable, pour peu que l'on ait plusieurs messages chiffrés, en utilisant des méthodes statistiques (on utilise le fait que certaines lettres de l'alphabet sont plus utilisées que d'autres dans la langue utilisée).

7.8.3.) ENIGMA

Enigma est une machine à rotors allemande qui fut utilisée durant la seconde guerre mondiale. Le code que sa version de base engendrait a été "cassé" par le "Chiffre" polonais (regroupement militaire des cryptanalystes de Pologne) grâce à des renseignements des services secrets français. Malheureusement, entre temps, les allemands avaient fabriqué une machine plus performante que les polonais n'ont pas réussi à "casser". C'est pourquoi, ce sont les services du "Chiffre" britannique (comprenant Turing) qui s'en sont chargés (en utilisant la méthode du mot probable) grâce à une autre machine appelée alors "Bombe".

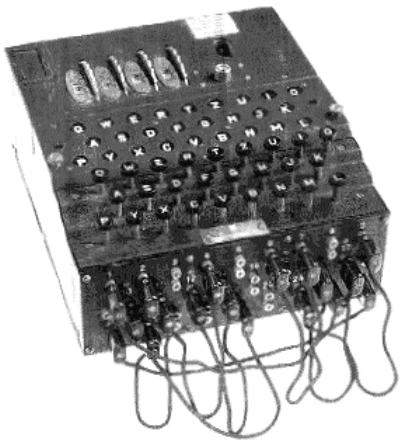
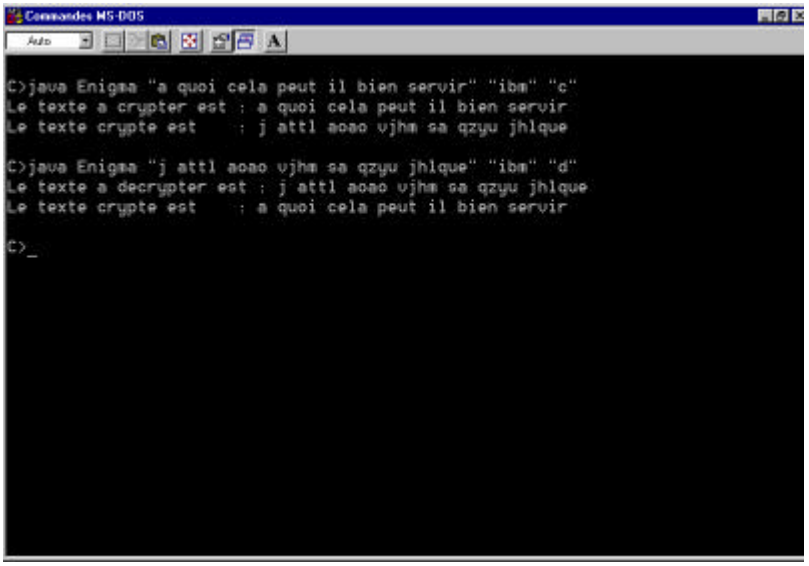


Figure 1 La machine Enigma

La machine Enigma utilisée par les allemands était composée de 5 éléments:

- Un tableau de connexions qui attribue à chaque lettre de l'alphabet une autre lettre de l'alphabet.
- 3 rotors qui établissent des connexions entre différentes lettres de l'alphabet et effectuent des rotations (d'où le problème du décryptage si on ne connaît pas leur position d'origine). En effet, dès qu'une lettre a été codée, le premier rotor tourne d'un cran; le second rotor tourne d'un cran chaque fois que le premier a fait un tour et le troisième rotor avance d'un cran quand le deuxième a fait un tour. Ce qui, compte tenu du nombre de combinaisons, équivaut à un codage de Vigenere avec une clef d'une période de 17576 (26*26*26).
- Le réflecteur quant à lui effectue des permutations entre les lettres et renvoie la lettre à coder une nouvelle fois à travers les 3 rotors.

Voici l'utilisation du programme « Enigma » :



Et voici le code source d' « Enigma » :

```
public class Enigma {

public static void main(String[] argv) {

    Rotor r1,r2,r3,connexions,réflecteur;
    String chaine_a_crypter = " ";
    String chaine_cryptee = new String();
    String cle = "abc";
    char c;
    boolean decodage = false;

    // Initialisation des rotors
    connexions = new Rotor("ekmflgdqvzntowyhxuspaibrcj");
    réflecteur = new Rotor("yruhqsldpxngokmiebfzcwvjat");
    r1 = new Rotor("ajdksiruxblhwtmcqgznpyfvoe");
    r2 = new Rotor("bdfhjlcprtxvznyeiwgakmusqo");
    r3 = new Rotor("esovpzjayquirhxlnftgkdcmwb");

    // On récupère les arguments
    if (argv.length > 0)
    chaine_a_crypter = argv[0]; else chaine_a_crypter = "La reine blanche
comme lys";

    chaine_a_crypter = chaine_a_crypter.toLowerCase();
    if (argv.length > 1)
          cle = argv[1]; else cle = "abc";

    if (argv.length > 1)
    {
          c = argv[2].charAt(0);
          if ((c=='d') | (c=='D')) decodage = true;
    }

    cle = cle+"abc";
    cle = cle.toLowerCase();
    cle = cle.substring(0,3);

    // On positionne les rotors
    r1.set_position(cle.charAt(0));
    r2.set_position(cle.charAt(1));
    r3.set_position(cle.charAt(2));
    // Cryptage de chaine_a_crypter
    for (int i=0;i<chaine_a_crypter.length();i++)
    {
          c = chaine_a_crypter.charAt(i);
          if ((c>='a') & (c<='z'))
          {
                // On passe à travers les rotors
                c = connexions.getchar(c,decodage);
                c = r1.getchar(c,decodage);
                c = r2.getchar(c,decodage);
                c = r3.getchar(c,decodage);
                // Deuxième passage en sens inverse
                c = réflecteur.getchar(c,decodage);
                c = r3.getchar(c,decodage);
                c = r2.getchar(c,decodage);
                c = r1.getchar(c,decodage);
                c = connexions.getchar(c,decodage);
                // On tourne les rotors
                if (r1.rotation())
                {
                      if (r2.rotation()) r3.rotation();
                }
          }
          chaine_cryptee = chaine_cryptee + c;
    }
```

```
    // On affiche les résultats
    if   (decodage   )   System.out.println("Le   texte   a   decrypter   est   :
"+chaine_a_crypter);
    else System.out.println("Le texte a crypter est : "+chaine_a_crypter);
    System.out.println("Le texte crypte est    : "+chaine_cryptee);
    }
}

class Rotor
{
    private String valeur;
    final String alphabet = new String ("abcdefghijklmnopqrstuvwxyz");
    int pos;
    Rotor()
    {
          valeur = "abcdefghijklmnopqrstuvwxyz";
          pos = 1;
          }
    Rotor( String s)
    {
          valeur = s;
          pos = 1;
    }
    public boolean rotation ()
    {
          String s;
          boolean ret = false;
          s = valeur.substring(0,1);
          valeur = valeur.substring(1,26)+s;
          pos ++;
          if (pos == 26)
          {
                pos = 1;
                ret = true;
          }
          return ret;
    }
    public void set_position( char p)
    {
          if ((p<'a') | (p>'z')) p='a';
          while (valeur.charAt(0) != p) rotation();
    }
    public char getchar( char c , boolean code)
    {
          int p = 0;
          if (code)
          {
                p = alphabet.indexOf(c);
                return valeur.charAt(p);
          }
          else
          {
                p = valeur.indexOf(c);
                return alphabet.charAt(p);
          }
    }
}
```

L'avantage principal de cette méthode est que, comme nous l'avons vu, elle est similaire à la méthode de Vigenere avec une clef d'une période de 17576. D'où la difficulté de déchiffrer. Cependant, il possède aussi la même faiblesse que la méthode de Vigenere, c'est à dire qu'il peut être vaincu par une "attaque statistique" (en supposant, comme c'était le cas pour le Chiffre polonais, que nous n'ignorons pas tout du fonctionnement de la machine).

8.) Exercices.

EXERCICE 3.1 :

Ecrire et tester un programme permettant de décoder des phrases codées avec le programme « César ».

Utilisez ce programme pour décoder cette « fable-express » anonyme :

> *Un jeune enfant, sur son pot, s'efforçait.*
> ***Moralité :***
> *NGRGVKVRQWUUCKV*

EXERCICE 3.2 :

Ecrire et tester un programme permettant de décoder des phrases codées avec le programme « Vigenere ».

Utilisez ce programme pour décoder cette « fable-express » d'Eugène Chavette :

> *Pépin le Bref est mort depuis bientôt mille ans.*
> ***Moralité :***
> *PFFTHLECNGDHOGHCNGPHLICHSTTCTKS*[1]

EXERCICE 3.1 :

Il existe, dans certains systèmes Unix, un utilitaire appelé *ROT13* permettant de coder un fichier en utilisant la même technique que l'algorithme « César ».La différence est que *ROT13* décale les caractères de 13 positions : il remplace le 'A' par le 'N', le 'B' par le 'O', le 'C' par le 'P' etc... Les caractères qui ne sont pas compris entre 'A' et 'Z' restent inchangés.

Quelle propriété particulière possède cet utilitaire ?

[1] La clé est « VARI ».