R & T 1
Algorithmique et programmation en Java

Parsy Jean-Pierre

2009-2010

- Variables élémentaires
- Actions primitives

Une variable est caractérisée par :

- son nom
- son type
- sa valeur

Nom d'une variable

- Le nom d'une variable est appelé identificateur.
- Un identificateur est une suite de caractères alphanumériques commençant par une lettre.
- Un identificateur ne comporte pas de caractère espace.
- On distingue les lettres majuscules et minuscules.
- Par convention, en java le nom d'une variable commence par une lettre minuscule.
- Par souci de lisibilité, il est important de donner aux variables des noms significatifs.
- Le nom d'une variable ne peut pas être égal à un mot réservé du langage, par exemple `for`, `if`, `int` ...

Type d'une variable

Le type d'une variable précise :

- l'ensemble des valeurs possibles.
- et l'ensemble des opérations permises.

Types prédéfinis

Les types prédéfinis permettent de manipuler :

- les nombres entiers de type int
- les nombres réels de type double
- les booléens de type boolean
- les caractères de type char
- les chaînes de caractères de type String

Valeurs d'une variable

Les valeurs d'une variable sont représentées par des littéraux :
Par exemple

1. int

   `152 -12 1`

2. double

   `0.5 1.0 1e-6 -2.7e3`

3. boolean

   `true false`

4. char

   `'A' '1'`

5. String

   `"RT & 1" "1" "A" ""`

Opérations sur les entiers

Les opérations possibles sont :

- $+$ addition
- $-$ soustraction
- $*$ multiplication
- $/$ division entière
- % reste de la division entière

Par exemple

```
17/5 est égal à 3
17%5 est égal à 2

car    17 = 5 * 3 + 2
```

Opérations sur les réels

Les opérations possibles sont :

- $+$ addition
- $-$ soustraction
- $*$ multiplication
- $/$ division

Par exemple

```
17./5.0 est égal à 3.4
17.0/5 est égal à 3.4
17/5.0 est égal à 3.4
```

les opérations sur les booléens et les chaînes de caractères seront vues ultérieurement

Déclaration d'une variable

- Une variable doit être déclarée.
- La déclaration d'une variable est de la forme :

  type nomVar ;
- Par exemple :
  - int i ;
  - double x ;
  - boolean b ;
  - char c ;
  - String ch ;

Actions primives

Les actions primitives permettent de réaliser :

- la saisie ou l'entrée de données à partir du clavier
- le traitement ou calcul et la mémorisation du résultat
- l'affichage ou la sortie de données à l'écran

Saisie ou entrée à partir du clavier

- Pour saisir la valeur d'une variable entière i

```
i = EntreeClavier . readInt ( "i ? " ) ;
```

- Pour saisir la valeur d'une variable réelle x

```
x = EntreeClavier . readDouble ( "x ? " ) ;
```

- Pour saisir la valeur d'une variable chaîne de caractères ch

```
ch = EntreeClavier . readString ( "ch ? " ) ;
```

Affectation

Un calcul ou traitement suivi de la mémorisation du résultat est réalisée par l'affectation
L'affectation

var = expr ;

permet de donner la valeur de l'expression `expr` à la variable `var`
Cette affectation réalise deux traitements

1. calcul de la valeur de l'expression `expr`
2. rangement de cete valeur dans la variable `var`

Affichage ou sortie à l'écran

Pour afficher la valeur d'une variable `var`

```
System.out.println ( var ) ;
```

Pour indiquer le nom de la variable `var`

```
System.out.println ( "valeur de var = "  + var ) ;
```

voir les expressions arithmètiques et les expressions booléennes

lancez eclipse en tapant dans un terminal eclipse
patientez c'est un peu long
insérez le projet /opt/shareJava/I3-PROJET.tar.gz dans l'espace de tavail d'eclipse
pour cela File/Import General Existing projets into workspace next
cochez la case select archive file
choisir le fichier /opt/shareJava/I3-PROJET.tar.gz

ouvrir le fichier `TestProjet.java`
pour l'exécuter clic droit sur le fichier Run As/ Java Application
modifier le message affiché

Ecrire un programme `Essai_1.java`
Pour cela dans le paquetage tp1 bouton droit new class
donner le nom Essai_1
décocher la case Inherited abstract methods
cocher la case public static void main (String [] args)
dans la méthode main écrire le code Java qui permet

- de déclarer une variable a de type entier
- de lui donner la valeur 5
- d'afficher la valeur de la variable x
- constater que le résultat affiché est correct

avant l'affichage insérer l'instruction $x = x + 1;$
justifier le résultat obtenu

Calcul de la surface et du périmètre d'un cercle

Calcul de la moyenne de trois nombres entiers

Echange des valeurs de deux variables entières

Conversion degrés radians

Conversion nombre de secondes en heures,secondes et minutes

Les opérations sur les booléens

Les opérateurs possibles sont :

- ! non
- && et court-circuit
- & et non court-circuit
- || ou court-circuit
- | ou non court-circuit

L'opérateur non

p	! p
false	true
true	false

L'opérateur et non court-circuit

p	q	p & q
false	false	false
false	true	false
true	false	false
true	true	true

L'opérateur et court-circuit

p	q	p && q
false	?	false
true	false	false
	true	true

Si le premier opérande **p** est faux le deuxième opérande **q** n'est pas évalué (il peut être indéfini)

L'opérateur ou non court-circuit

p	q	p | q
false	false	false
false	true	true
true	false	true
true	true	true

L'opérateur ou court-circuit

p	q	p || q
true	?	true
false	false	false
	true	true

Si le premier opérande **p** est vrai le deuxième opérande **q** n'est pas évalué (il peut être indéfini)

Expressions booléennes
Opérateurs relationnels

==	égal
!=	différent
>	supérieur
<	inférieur
>=	supérieur ou égal
<=	inférieur ou égal

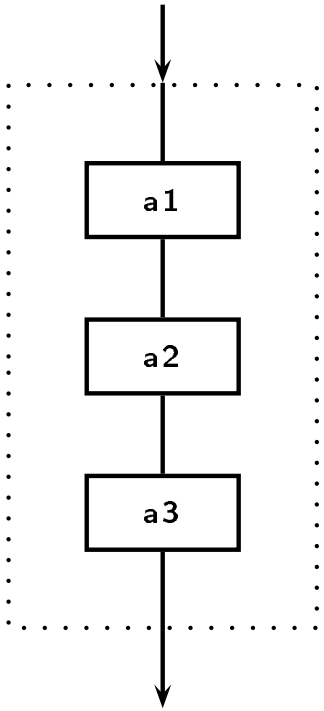
Structures de contrôle

Séquence

Alternative

Répétition

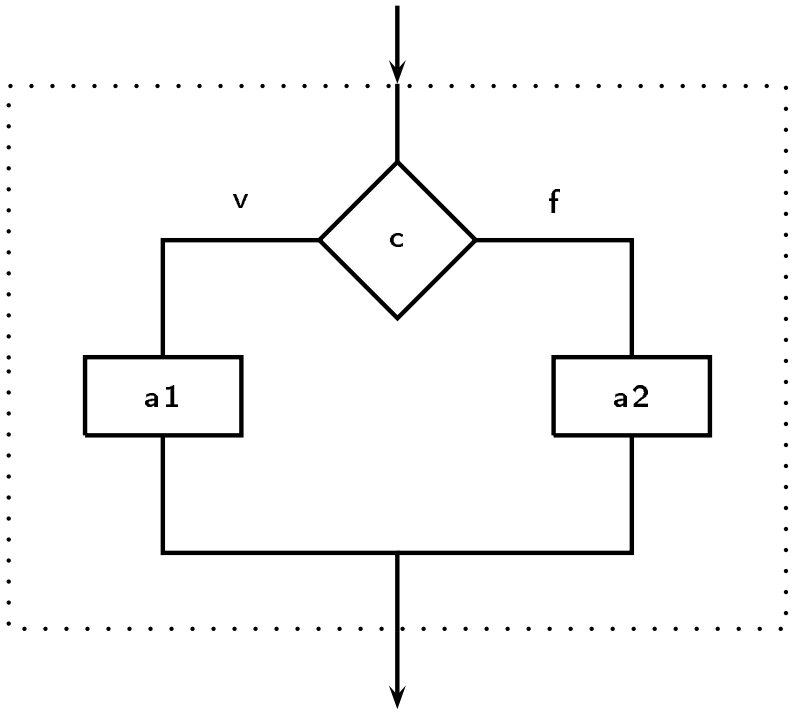
Séquence




```
a1
a2
a3
```

Les actions **a1**, **a2** et **a3** sont exécutées en suivant

Alternative



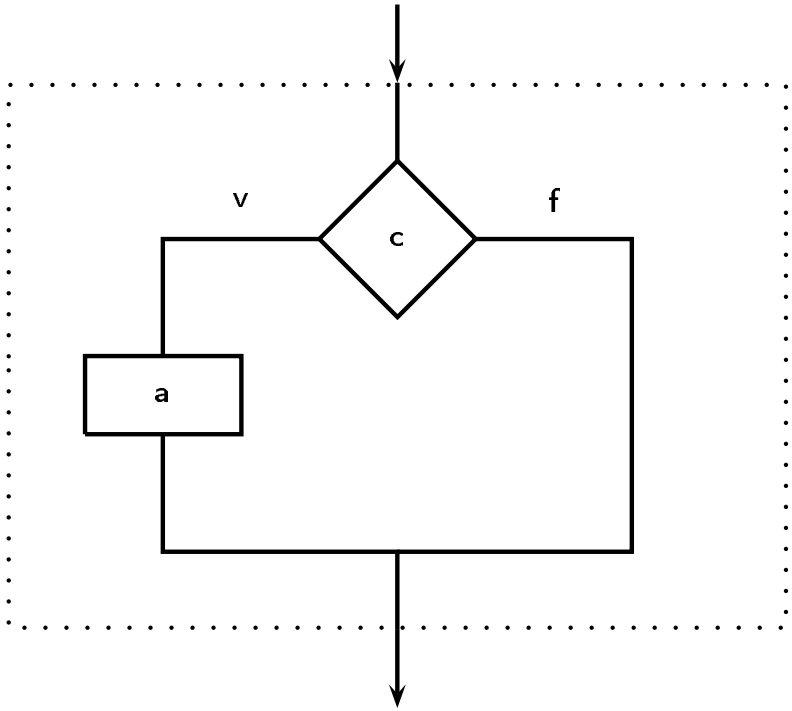

```
if ( c ) {
    a1
} else {
    a2
}
```

La condition **c** doit être une expression booléenne (dont la valeur est égale à vrai ou faux)
Si la condition est vraie, l'action **a1** est exécutée, dans le cas contraire l'action **a2** est exécutée

Alternative Exemple

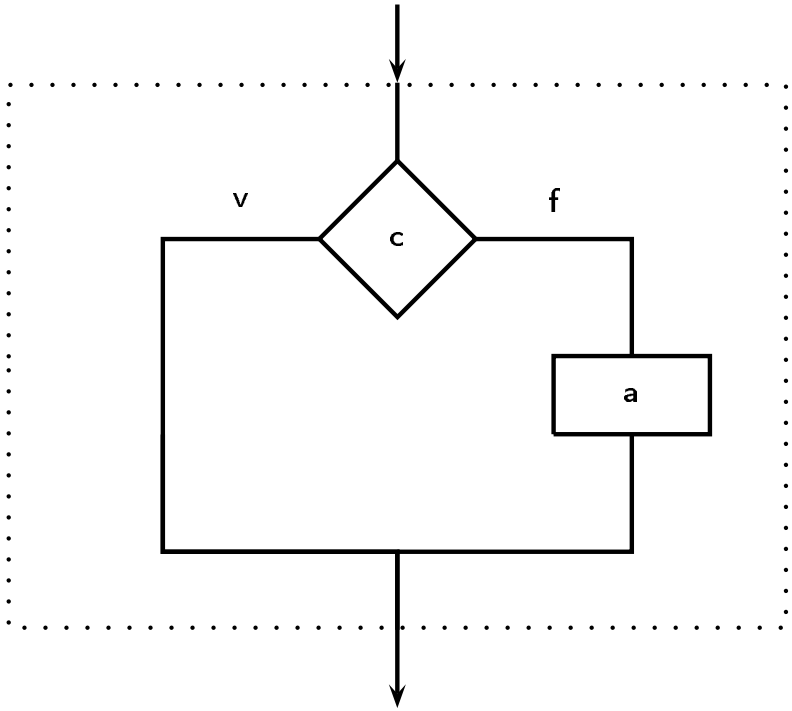
```
if ( qte >= 10 ) {
    fraisPort = 10 ;
}else{
    fraisPort = 50;
}
```

La partie sinon peut être vide




```
if ( c ) {
    a
}
```

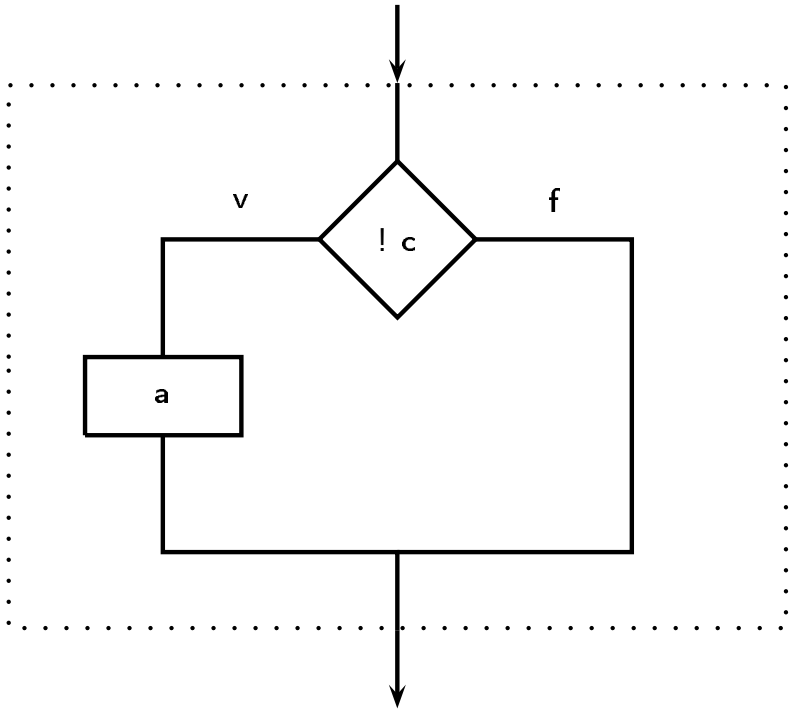
La partie alors peut être vide




```
if ( c ) {

} else {
    a
}
```

La partie alors peut être vide (suite)

Dans ce cas il est préférable de transformer l'alternative sous la forme suivante :




```
if ( ! c ) {
    a
}
```

Répétitions

Répétitions dont le nombre est connu la boucle **pour**

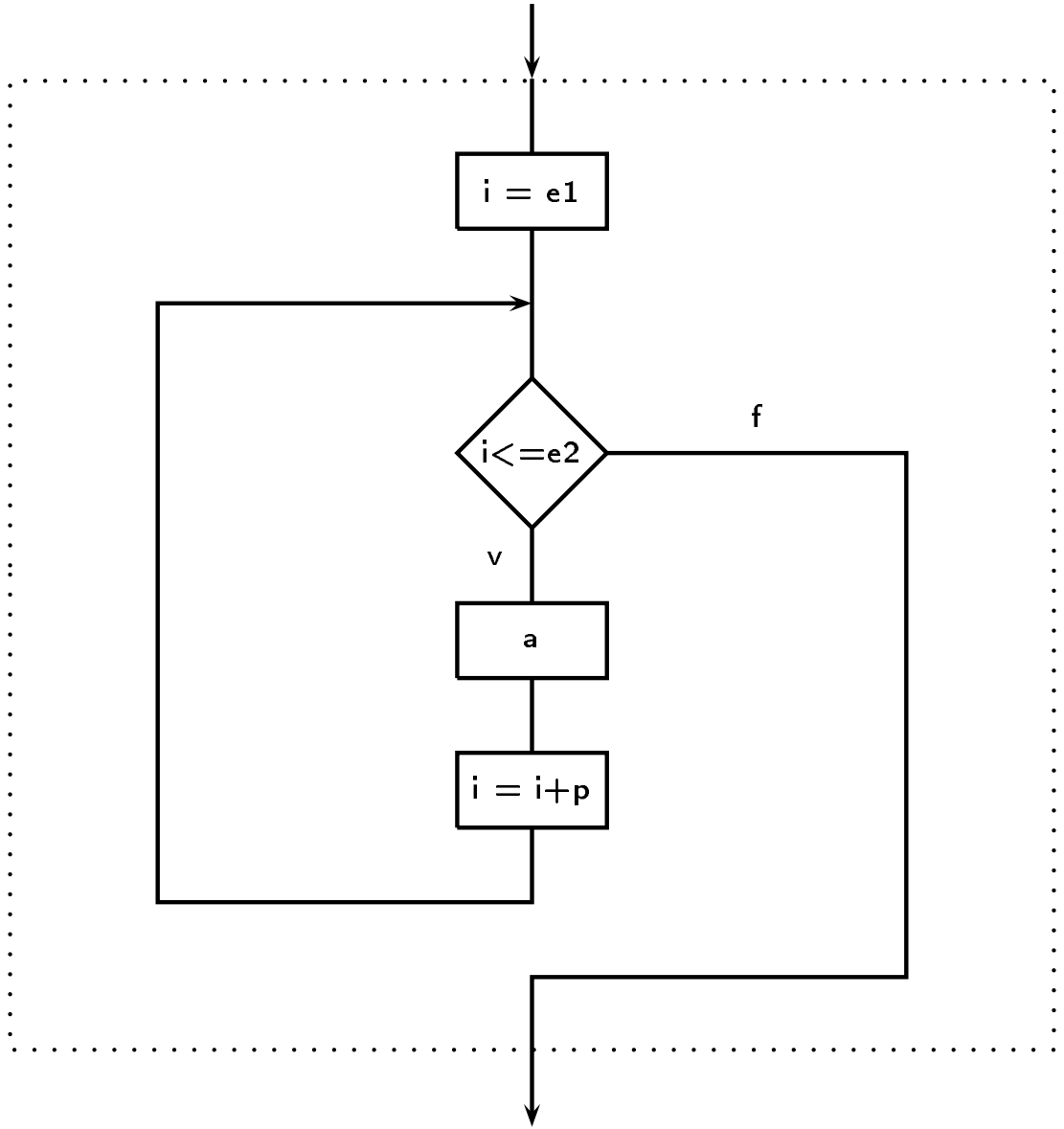
Répétitions

Répétitions dont le nombre est connu la boucle **pour**

La boucle pour avec un pas positif p

```
pour i de e1 à e2 pas p faire
    a
finpour
```

Répétition pour avec un pas positif p



Répétition pour avec un pas positif p

La variable **i** dite variable de contrôle

- est initialisée avec la valeur de **e1**
- si elle est inférieure ou égale à celle de **e2**
  - l'action **a** est exécutée
  - on ajoute ensuite le pas **p** à **i**
  - puis enfin on retourne à la comparaison avec **e2**
- si elle est supérieure à celle de **e2** on sort de la boucle pour

Remarque

- si la valeur de **e1** est supérieure à celle de **e2** l'action **a** n'est pas exécutée.

Répétition pour avec un pas p positif

```
for ( int i = e1 ; i <= e2 ; i = i + p) {

    a

}
```

Exercice 1

Afficher une ligne de 3 étoiles

Afficher une ligne de 5 étoiles

```
for ( int i = 1 ; i <= 5 ; i = i + 1) {
    System.out.print ( "*" ) ;
}
System.out.println ( ) ;
```

Exercice 2

Calcul de la somme des entiers de 1 à 10

$$1 + 2 + 3 + \cdots + 10$$

Calcul de la somme des entiers de 1 à 10

Calcul de la somme des entiers de 1 à 10

```
int som ;
som = 0 ;
for ( int i = 1 ; i <= 10 ; i = i + 1) {
    som = som + i ;
}
System.out.println ( "som = " + som ) ;
```

Exercice 3

Calcul de factoriel n

$$n! = 1 * 2 * 3 + \cdots * (n-1) * n$$

Exercice 3 Calcul de $n!$

On suppose que la variable entière **n** a été déclarée, saisie à partir du clavier et que sa valeur est positive ou nulle.

```
int fact ;
fact = 1 ;
for ( int i = 1 ; i <= n ; i = i + 1) {
    fact = fact * i ;
}
System.out.println ( "fact = " + fact ) ;
```

Exercice 4

Calcul de x^n avex **x** réel et **n** entier positif ou null

$$x^n = \underbrace{x * x * \cdots * x}_{n \ fois}$$

Exercice 4 Calcul de x^n

On suppose que la variable entière **n** a été déclarée, saisie à partir du clavier et que sa valeur est positive ou nulle.

On suppose que la variable réelle **x** a été déclarée et saisie à partir du clavier.

```
double puiss ;
puiss = 1 ;
for ( int i = 1 ; i <= n ; i = i + 1) {
    puiss = puiss * x ;
}
System.out.println ( "puiss = " + puiss ) ;
```

Exercice 5

Afficher un rectangle de 5 lignes de 10 étoiles

****

****

****

****

****

```
for ( int i = 1 ; i <= 5 ; i = i + 1 ) {
    for ( int j = 1 ; j <= 10 ; j = j + 1 ) {
        System.out.print ( "*" ) ;
    }
    System.out.println (  ) ;
}
```

Exercice 6

Calcul de la somme des entiers pairs compris entre 1 et 100.

$$2 + 4 + \cdots + 98 + 100$$

Exercice 7

Calcul de la somme des entiers impairs compris entre 1 et 100.

$$1 + 3 + \cdots + 97 + 99$$

Exercice 6 Calcul de la somme des entiers pairs compris entre 1 et 100.

```
int som ;
som = 0 ;
for ( int i = 2 ; i <= 100 ; i = i + 2) {
    som = som + i ;
}
System.out.println ( "som = " + som ) ;
```

Exercice 7 Calcul de la somme des entiers impairs compris entre 1 et 100.

```
int som ;
som = 0 ;
for ( int i = 1 ; i <= 100 ; i = i + 2) {
    som = som + i ;
}
System.out.println ( "som = " + som ) ;
```

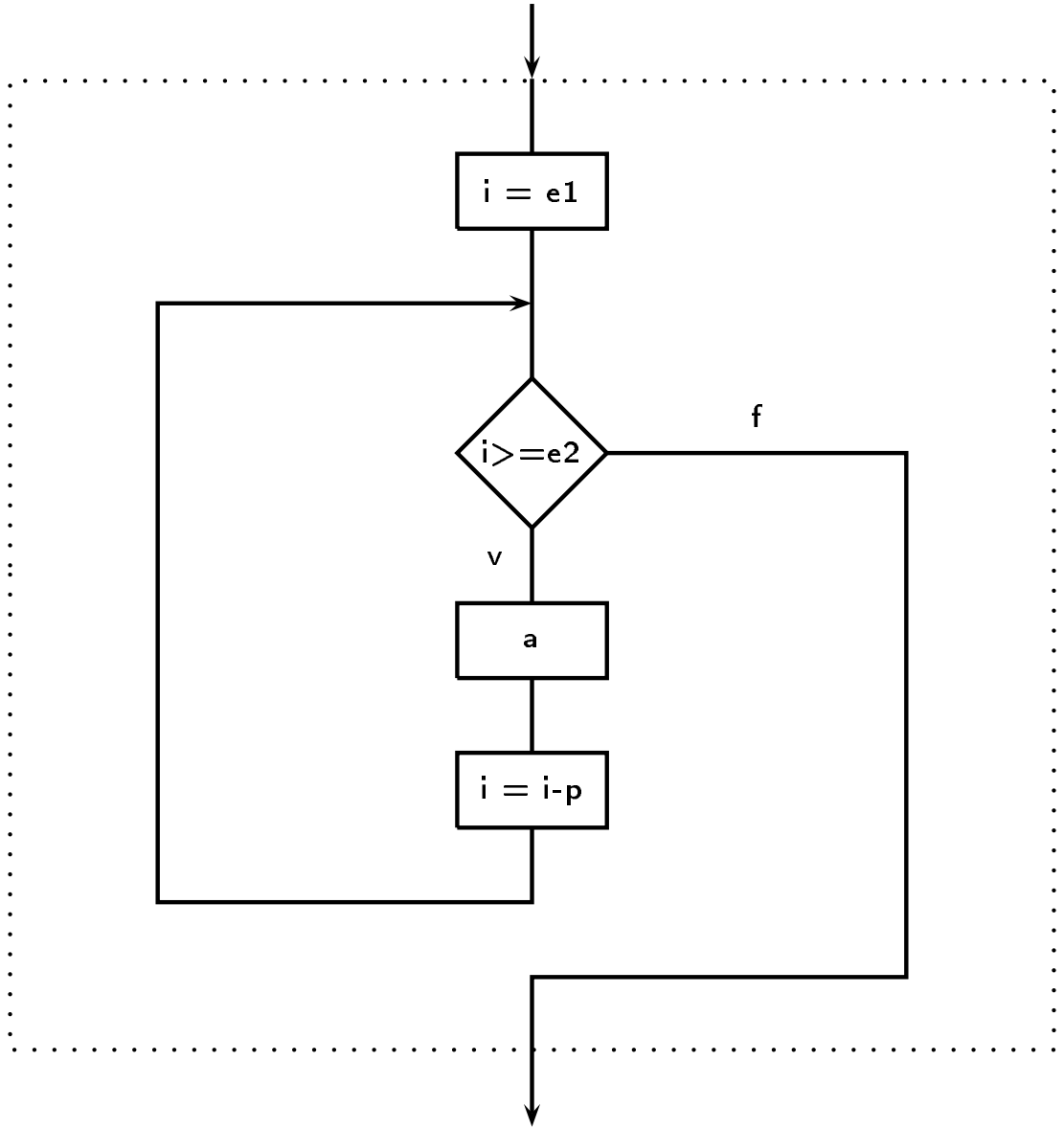
Répétitions

Répétitions dont le nombre est connu la boucle **pour**

La boucle pour avec un pas négatif -p

```
pour i de e1 à e2 pas -p faire
    a
finpour
```

Répétition pour avec un pas négatif -p



Répétition pour avec un pas négatif -p

La variable **i** dite variable de contrôle

- est initialisée avec la valeur de **e1**
- si elle est supérieure ou égale à celle de **e2**
  - l'action **a** est exécutée
  - on soustrait ensuite le pas **p** à **i**
  - puis enfin on retourne à la comparaison avec **e2**
- si elle est inférieure à celle de **e2** on sort de la boucle pour

Remarque

- si la valeur de **e1** est inférieure à celle de **e2** l'action **a** n'est pas exécutée.

Répétition pour avec un pas p négatif

```
for ( int i = e1 ; i >= e2 ; i = i - p) {

    a

}
```

Exercice 8

Quels sont les résutats affichés par le programme suivant.

```
for ( int i = 1 ; i <= 10 ; i = i + 2) {
    System.out.println ( i ) ;
}
```

Exercice 9

Quels sont les résutats affichés par le programme suivant.

```
for ( int i = 10 ; i >= 1 ; i = i - 2) {
    System.out.println ( i ) ;
}
```

Exercice 10 Quels sont les résutats affichés par le programme suivant.

```
int n ;
n = 150 ;
int p ;
p = 128 ;
for ( int i = 1 ; i <= 8 ; i = i + 1) {
      if ( n >= p ) {
            System.out.print ( "1" ) ;
            n =  n - p ;
      } else {
            System.out.print ( "0" ) ;
      }
      p = p / 2 ;
}
System.out.println ( ) ;
```

Les fonctions

Une fonction réalise un traitement particulier qui être utilisé plusieurs fois

- dans le même programme (classe)
- ou dans un autre programme.

Les fonctions (2)

L'utilisation des fonctions améliore la lisibilité d'un programme.

Le corps de la fonction **main** qui ne doit pas dépasser une vingtaine de lignes (un écran) est une suite d'appels de fonctions définies dans le même ou un autre programme.

Les fonctions (3)

Une fonction est un boîte noire qui réalise un traitement particulier qu'on peut utiliser sans connaître exactement les instructions qui constituent le corps de la fonction.

Vous avez déjà utilisé :

- les fonctions mathématiques de la classe **Math**
- les fonctions de saisie de la classe **EntreeClavier**

sans connaître précisément comment sont réalisées ces fonctions.

Déclaration ou définition d'une fonction

La déclaration d'une fonction (avant la fonction main) est de la forme :

```
public static typeValRet nomFct ( listParamForm ) {

    // corps de la fonction

}
```

- *typeValRet* est le type de la valeur retournée.
  - *void* indique que la fonction ne retourne rien
- le nom de la fonction *nomFct* doit indiquer clairement le traitement effectuée par la fonction.
- *listParamForm* est la liste des paramètres formels

Appel d'une fonction

L'appel d'une fonction (dans le main) est de la forme

- Dans le cas où la fonction est définie dans la classe où elle utilisée.

```
nomFct ( listeParamEff ) ;
```

- Dans le cas contraire :

```
NomClasse.nomFct ( listeParamEff ) ;
```

  - *listParamEff* est la liste des paramètres effectifs.
  - *NomClasse* est le nom de la classe où est définie la fonction.

Les parenthèses sont obligatoires même si la liste des paramètres effectifs est vide.

Définition de la fonction hello

La fonction **hello** affiche le message Bonjour.

```
public static void hello ( ) {
    System.out.println ( "Bonjour" ) ;
}
```

Appel de la fonction hello

Le programme suivant :

```
public class Test1 {

    public static void hello ( ) {
        System.out.println ( "Bonjour" ) ;
    }

    public static void main ( String [ ] args ) {
        for ( int i = 1 ; i <= 5 ; i = i + 1 ) {
            hello ( ) ;
        }
    }
}
```

affiche 5 fois Bonjour

Définition de la fonction afficheMsg

La fonction **afficheMsg** affiche le message **msg** passé en paramètre.

```
public static void afficheMsg ( String msg ) {

    System.out.println ( msg ) ;

}
```

Appel de la fonction afficheMsg

Le programme suivant :

```
public class Test2 {
    public static void afficheMsg ( String msg  ) {
        System.out.println ( msg ) ;
    }

    public static void main ( String [ ] args ) {
        afficheMsg ( "Bonjour" ) ;
        afficheMsg ( "Bonsoir" ) ;
    }
}
```

affiche Bonjour puis Bonsoir

Les fonctions carré et cube

Ces deux fonctions retournent un entier.

```
public static int carre ( int n ) {
    return n * n ;
}

public static int cube ( int n ) {
    return carre ( n ) * n ;
}
```

Exemple

Quel est le résultat du programme suivant ?

```
public class Test3 {

    public static int carre ( int n ) {
        return n * n ;
    }

    public static void main ( String [ ] args ) {
        carre ( 5 ) ;
    }
}
```

Exemple

La valeur retournée par la fonction **carre** doit être mémorisée dans une variable

```
public class Test3 {

    public static int carre ( int n ) {
        return n * n ;
    }

    public static void main ( String [ ] args ) {
        int result ;
        result = carre ( 5 ) ;
        System.out.println(result);
    }
}
```

Exemple

ou passée en paramètre à une autre fonction

```
public class Test3 {

    public static int carre ( int n ) {
        return n * n ;
    }

    public static void main ( String [ ] args ) {
        System.out.println(carre(5));
    }
}
```

Exercice 1

Donner le programme qui permet d'obtenir la table des carrés et des cubes des nombres entiers de 1 à 10.

1	1	1
2	4	8
3	9	27
4	16	64
5	23	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

en utilisant les fonctions carre et cube précédentes.

Exercice 1

```
public class Exercice1 {
    public static int carre ( int n ) {
        return n * n ;
    }
    public static int cube ( int n ) {
        return carre ( n ) * n ;
    }
    public static void main ( String [ ] args ) {


    }
}
```

Exercice 2

Reprendre l'exercice bonjourPersonnel sous forme d'une fonction.

```
bonjourPersonnel ( "Duchmol" , 'M' ) ;
```

```
affiche Bonjour Monsieur Duchmol
```

```
bonjourPersonnel ( "Duchmol" , 'F' ) ;
```

```
affiche Bonjour Madame Duchmol
```

Exercice 2

```
public class Exercice2 {
    public static void bonjourPersonnel
                        ( String nom , char sexe ) {


    }
    public static void main ( String [ ] args ) {


    }
}
```

Exercice 3

Ecrire la fonction **max2** qui retourne le maximum de ses deux paramètres entiers **a** et **b**.

Exercice 4

Ecrire la fonction **max3** qui retourne le maximum de ses trois paramètres entiers **a**, **b** et **c** en utilisant le fonction **max2**.

Exercice 5

Ecrire la fonction **som1** qui retourne la somme des entiers de 1 à n.

$$1 + 2 + 3 + 4 + \ldots + n$$

Exercice 6

Ecrire la fonction **som2** qui retourne la somme des carres de entiers de 1 à n.

$$1 + 2^2 + 3^2 + 4^2 + \ldots + n^2$$

Exercice 7

Ecrire la fonction **som1Bis** qui retourne la somme alternée des entiers de 1 à n.

$$1 - 2 + 3 - 4 + \ldots + (-1)^{n+1} n$$

Exercice 6

Ecrire la fonction **som2Bis** qui retourne la somme des carres de entiers de 1 à n.

$$1 - 2^2 + 3^2 - 4^2 + \ldots + (-1)^{n+1} n^2$$

Exercice 7

Ecrire la fonction **estChiffreDec** qui retourne **true** si la caractère **c** passé en paramètre est un chiffre décimal et **false** dans le cas contraire.

Les tableaux à une dimension (ou vecteur)

Un tableau est un groupement d'éléments de même type.

Un tableau permet de représenter plusieurs variables sous le même nom.

Les éléments d'un même tableau seront distingués à partir de leur rang ou indice.

Exemple 1 (1)

Un tableau **a** de 8 entiers indicés de 0 à 7.

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

L'élémént d'indice i est noté a [i].

Par exemple a [3] est égal à 20.

Exemple 1(2)

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

L'instruction

```
a [ 4 ] = 55 ;
```

permet de donner la valeur 55 à l'élément d'indice 4 du tableau a

	0	1	2	3	4	5	6	7
a	10	50	-3	20	55	0	1	2

Question 1

Donner l'instruction qui permet d'ajouter 1 à l'élément d'indice 3 du tableau **a**.

Les tableaux nombre d'éléments

Les éléments d'un tableau sont tous de même type

- entiers
- réels
- caractères
- ...

Les élements sont indicés à partir de 0.

Le nombre d'éléments du tableau **a** est `a.length`

L'indice du dernier élément du tableau **a** est `a.length - 1`

Les tableaux déclaration et construction ou initialisation

Un tableau doit être

- déclaré
- puis construit ou initialisé.

Les tableaux déclaration

La déclaration d'un tableau est de la forme

```
typeEltsTab [ ] nomTableau ;
```

- **typeEltsTab** est le type des éléments du tableau.
- **nomTableau** est le nom du tableau.
- les crochets indiquent que l'on déclare un tableau.

  Si on les oublie on déclare une variable simple.

Les tableaux déclaration exemples

Déclaration d'un tableau **a** d'entiers :

```
int [ ] a ;
```

Déclaration d'un tableau **u** de réels :

```
double [ ] u ;
```

Déclaration d'un tableau **tabbool** de booléen :

```
boolean [ ] tabbool ;
```

Déclaration d'un tableau **voyelles** de caractères :

```
char [ ] voyelles ;
```

Déclaration d'un tableau **nomsMois** de chaînes de caractères :

```
String [ ] nomMois ;
```

Les tableaux construction

La construction d'un tableau est de la forme :

```
nomTableau = new typeEltsTab [ nbEltsTab ] ;
```

- **nomTableau** est le nom du tableau.
- **typeEltsTab** est le type des éléments du tableau.
- **nbEltsTab** est le nombre d'éléments du tableau.

  C'est une expression dont la valeur est définie.

Les tableaux construction exemples

Construction du tableau **a** de 8 entiers :

```
a = new int [ 8 ] ;
```

Les tableaux construction exemples

Le nombre d'éléments du tableau peut être un expression.

Construction du tableau **u** de n réels :

```
int n ;
n = EntreeClavier . readInt ( "n ? " ) ;

u = new int [ n ] ;
```

Tableaux initialisation

L'initialisation qui permet de construire un tableau en donnant des valeurs à ses éléments est de la forme :

```
nomTab = new typeEltsTab [ ]

                { val1 , val2 , ... , valn } ;
```

Tableaux initialisation exemples (1)

Les valeurs du tableau **a** :

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

sont données par l'instruction :

```
a = new int [ ] { 10,50,-3,20,-8,0,1,2 } ;
```

Tableaux initialisation exemples (2)

L'initialisation du tableau de caractères **voyelles** est donnée par l'instruction :

```
voyelles = new char [ ] { 'a','e','i','o','u','y' } ;
```

Tableaux initialisation exemples (3)

L'initialisation du tableau de chaînes de caractères **nomsMois** est donnée par l'instruction :

```
nomsMois=new String [ ]  "Jan","Fév","Mar", ...  ;
```

Tableaux Exercice 1

Donner les instructions qui permettent d'ajouter 1 à tous les éléments du tableau **a**.

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

Le résultat doit être :

	0	1	2	3	4	5	6	7
a	11	51	-2	20	-7	1	2	2

Tableaux Exercice 1

```java
package tp6;

public class Exo1 {

    public static void main ( String [ ] args ) {

        int [  ] a ;

        a = new int[ ]{ 10, 50, -3, 20, -8, 0, 1, 2 } ;

        // ajouter 1  a tous les elements du tableau

        // afficher les elts de a sur seule ligne
    }
}
```

Tableaux Exercice 2

Donner les instructions qui permettent calculer la somme des éléments du tableau **a**.

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

Tableaux Exercice 2

```java
package tp6;

public class Exo2 {

    public static void main ( String [ ] args ) {

        int [ ] a ;

        a = new int[ ]{ 10, 50, -3, 20, -8, 0, 1, 2 } ;

        // calculer la somme des elts de a

        // afficher la somme des elts de a
    }
}
```

Tableaux Exercice 3

Donner les instructions qui permettent d'échanger les éléments d'indice **i1** et **i2**

i1	3		i2	6

du tableau **a** :

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

Le résultat doit être :

	0	1	2	3	4	5	6	7
a	10	50	-3	1	-8	0	20	2

Tableaux Exercice 3

```java
package tp6;

public class Exo3 {

    public static void main ( String [ ] args ) {

        int [ ] a ;

        a = new int[ ]{ 10, 50, -3, 20, -8, 0, 1, 2 } ;

        // declarer et saisir i1 et i2

        // echanger les elts d'indice i1 et i2 de a

        // afficher les elts de a sur seule ligne

    }
}
```

Tableaux Exercice 4 Rotation circulaire gauche

Donner les instructions qui permettent d'effectuer une permutation circulaire gauche des éléments du tableau **a** :

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

Le résultat doit être :

	0	1	2	3	4	5	6	7
a	50	-3	20	-8	0	1	2	10

Tableaux Exercice 4 Rotation circulaire gauche

```
package tp6;

public class Exo4 {

    public static void main ( String [ ] args ) {

        int [  ] a ;

        a = new int[ ]{ 10, 50, -3, 20, -8, 0, 1, 2 } ;

        // rotation circulaire gauche de a

        // afficher les elts de a sur seule ligne

    }
}
```

Tableaux Exercice 5 Rotation circulaire droite

Donner les instructions qui permettent d'effectuer une permutation circulaire droite des éléments du tableau **a** :

	0	1	2	3	4	5	6	7
a	10	50	-3	20	-8	0	1	2

Le résultat doit être :

	0	1	2	3	4	5	6	7
a	2	10	50	-3	20	-8	0	1

Tableaux Exercice 5 Rotation circulaire droite

```
package tp6;

public class Exo5 {

    public static void main ( String [ ] args ) {

        int [  ] a ;

        a = new int[ ]{ 10, 50, -3, 20, -8, 0, 1, 2 } ;

        // rotation circulaire droite de a

        // afficher les elts de a sur seule ligne

    }
}
```

Modification des paramètres (1)

Soit la fonction **change1** :

```
public static void change1 ( int a ) {
    a = a + 10 ;
}
```

Quel est le résultat de l'exécution des instructions suivantes :

```
int b ;
b = 5 ;
System.out.println ( "b = " + b ) ;
change1 ( b ) ;
System.out.println ( "b = " + b ) ;
```

Modification des paramètres (2)

Soit la fonction **change2** :

```
public static int change2 ( int a ) {
    return a + 10 ;
}
```

Quel est le résultat de l'exécution des instructions suivantes :

```
int b ;
b = 5 ;
System.out.println ( "b = " + b ) ;
b = change2 ( b ) ;
System.out.println ( "b = " + b ) ;
```

Modification des paramètres (3)

Les paramètres d'une fonction qui ne sont pas des tableaux ne sont pas modifiés par cette dernière.

Modifications des paramètres tableaux

Une fonction peut modifier ses paramètres de type tableau.

Par exemple la fonction suivante :

```
public static void incr ( int [ ] a ) {
    for ( int i = 0 ; i <= a.length-1 ; i=i+1 ) {
            a [ i ] = a [ i ] + 1 ;
    }
}
```

permet d'ajouter 1 à tous les éléments du tableau passé en paramètre.

Modifications des paramètres tableaux (suite)

Que est le résulat de l'exécution des instructions suivantes :

```
int [ ] a ;
a = new int [ ] { 50 , 30 , 10 , 20 , 40 } ;
incr ( a ) ;
Tableaux . afficher ( a ) ;
```

Excercice 1

Ecrire la fonction **echange** qui permet d'échanger les éléments d'indice **i1** et **i2** du tableau **a**.

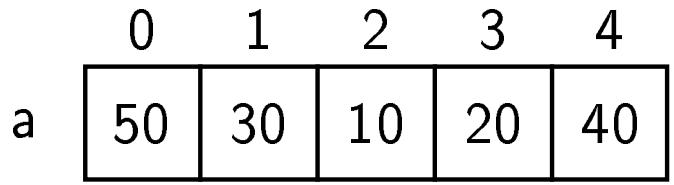
	0	1	2	3	4
a	50	30	10	20	40

Excercice 1 (suite)

```
package tp7 ;

public class Tri {
    public static void echange ( int [ ] a, int i1 , int i2 ) {


    }

    public static void main ( String [ ] args ) {
        int [ ] a ;
        a = new int [ ] { 50 , 30 , 10 , 20 , 40 } ;

        echange ( a , 2 , 1 ) ;

        Tableaux.afficher ( a );

    }
}
```

Excercice 1 (suite)

Ecrire la fonction **indMin** qui retourne l'indice du plus petit élément du tableau **a**.



indMin (a) retourne 2

Excercice 1 (suite)

```
package tp7 ;

public class Tri {
    public static void echange ( int [ ] a, int i1 , int i2 ) {


    }

    public static int indMin ( int [ ] a ) {

    }

    public static void main ( String [ ] args ) {
        int [ ] a ;
        a = new int [ ] { 50 , 30 , 10 , 20 , 40 } ;

        System.out.println ( indMax ( a ) ) ;


    }
}
```

Excercice 1 (suite)

Donner l'instruction qui permet d'échanger les éléments d'indice 0 et indMin(a) du tableau a.

	0	1	2	3	4
a	50	30	10	20	40

Après l'exécution de cette instruction on doit obtenir

	0	1	2	3	4
a	10	30	50	20	40

Ecrire le fonction **indMin** qui retourne l'indice du plus petit élément du sous-tableau **a** [**iDeb** : **a.length** − **1**]

a [**iDeb** : **a.length** − **1**] désigne le sous-tableau constitué des éléments de **a** dont les indices sont compris entre **iDeb** et **a.length** − **1**.

Excercice 1 (suite)

```
package tp7 ;

public class Tri {
    public static void echange ( int [ ] a, int i1 , int i2 ) {


    }

    public static int indMin ( int [ ] a ) {

    }

    public static int indMin ( int [ ] a , int iDeb ) {

    }

    public static void main ( String [ ] args ) {
        int [ ] a ;
        a = new int [ ] { 10 , 30 , 50 , 20 , 40 } ;

        System.out.println ( indMax ( a , 1 ) ) ;

    }
}
```

Pour le tableau

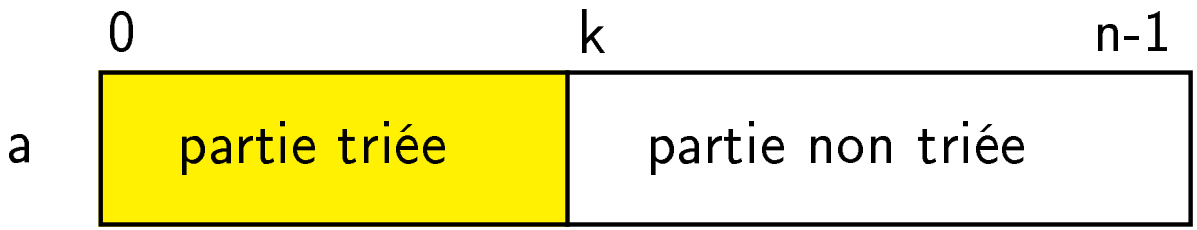
	0	1	2	3	4
a	10	30	50	20	40

```
indMin ( a , 1 ) retourne 3

indMin ( a ) retourne 0
```

Tri par sélection

Soit **a** un tableau d'entiers et **n** son nombre d'éléments.

n est égal à **a.length** − **1**



A l'étape **k** on recherche (séléctionne) le plus petit élément de la partie non triéé et on l'échange avec le premier élément (d'indice k) de cette partie non triée.

Tri par sélection (suite)

L'algorithme du tri par séléction est donc :

```
pour i de 0 à n - 2 faire

        // déterminer l'indice iMin du plus petit élément
        // de a [ k : n - 1 ]

        // échanger les éléments d'indice k et iMin

fpour
```

Tri par sélection (suite)

Pour le tableau a

	0	1	2	3	4
a	50	30	10	20	40

à l'étape **k=0** on trouve **iMin=2**, le tableau a devient :

	0	1	2	3	4
a	10	30	50	20	40

à l'étape **k=1** on trouve **iMin=3**, le tableau a devient :

Tri par sélection (suite)

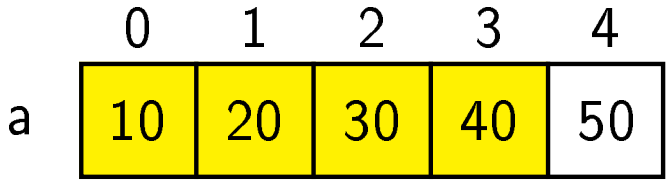
	0	1	2	3	4
a	10	20	50	30	40

à l'étape **k=2** on trouve **iMin=3**, le tableau a devient :

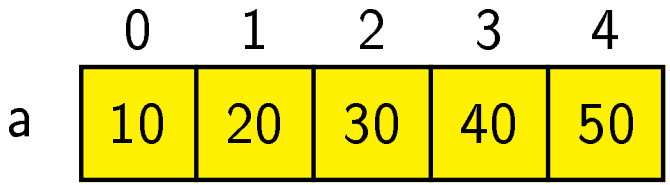
	0	1	2	3	4
a	10	20	30	50	40

à l'étape **k=3** on trouve **iMin=4**, le tableau a devient :

Tri par sélection (suite)



Il ne reste plus qu'un élément supérieur à tous ceux qui le précèdent, le tableau **a** est donc complètement trié.



Tri par sélection (suite)

Ecrire la fonction **tri** qui permet de trier par ordre croissant le tableau **a** passé en paramètre.

Cette fonction doit utiliser les deux fonctions **indMin** et **echange**

Excercice 1 (suite)

```
package tp7 ;

public class Tri {
    public static void echange ( int [ ] a, int i1 , int i2 ) {

    }

    public static int indMin ( int [ ] a ) {

    }

    public static int indMin ( int [ ] a , int iDeb ) {

    }

    public static void tri ( int [ ] ) {

    }

    public static void main ( String [ ] args ) {
        int [ ] a ;
        a = new int [ ] { 50 , 30 , 10 , 20 , 40 } ;
        tri ( a ) ;
        Tableaux.afficher( a );
    }
}
```

Tableau non rempli (1)

Le tableau **elts** contient 0 éléments, il pourra en contenir **elts.length-1**

nbElts donne le nombre d'éléments du tableau **elts**

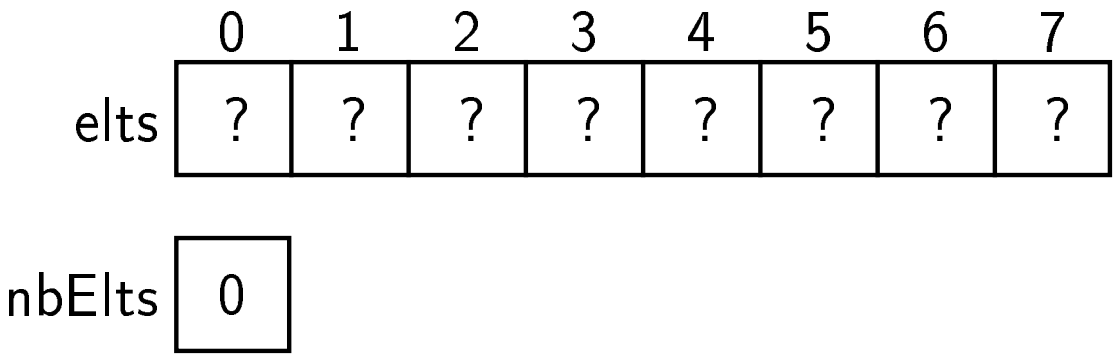
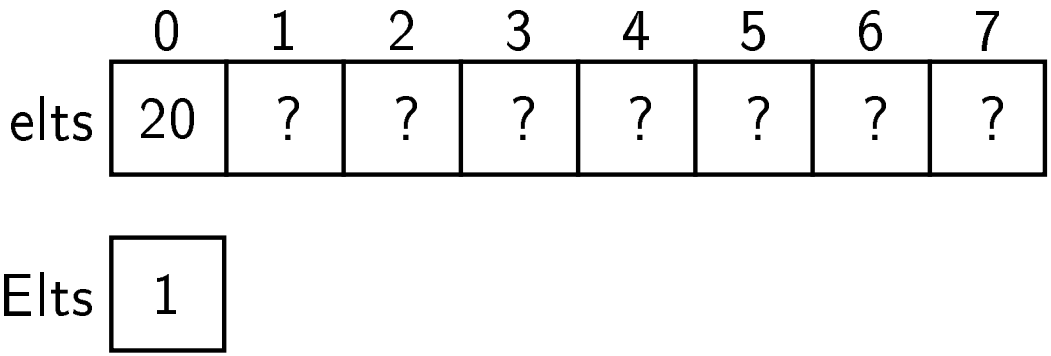


Tableau non rempli (2)

Aprés l'ajout de 20



Aprés l'ajout de 10

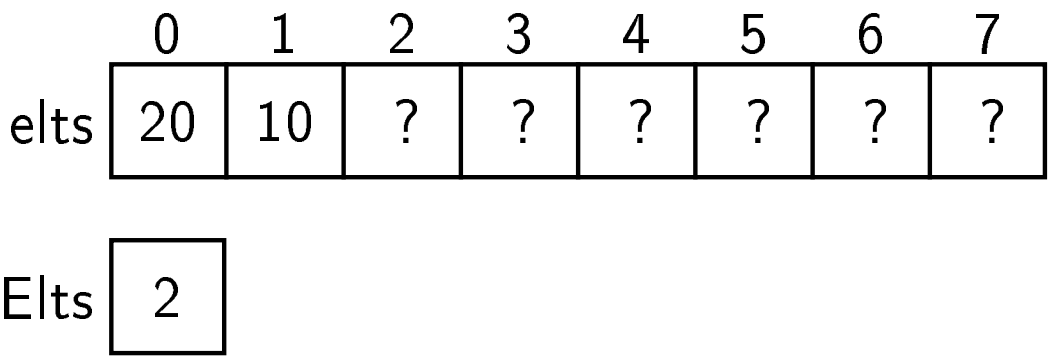
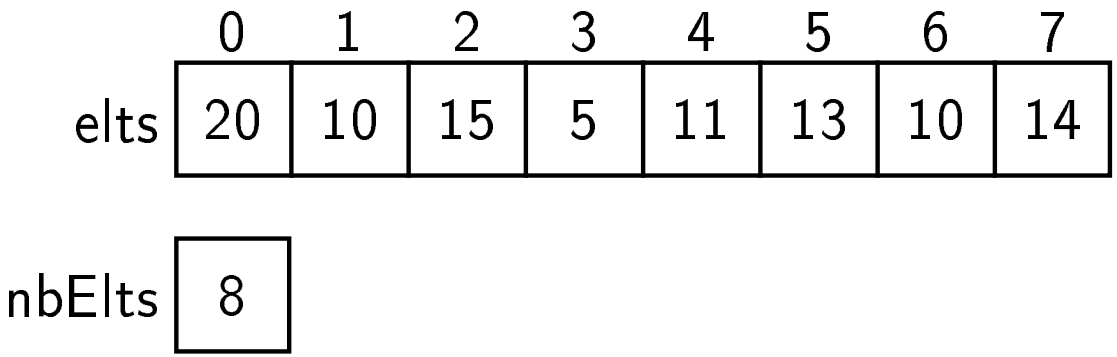


Tableau non rempli (3)

Aprés l'ajout de 15, 5, 11, 13, 10 et 14

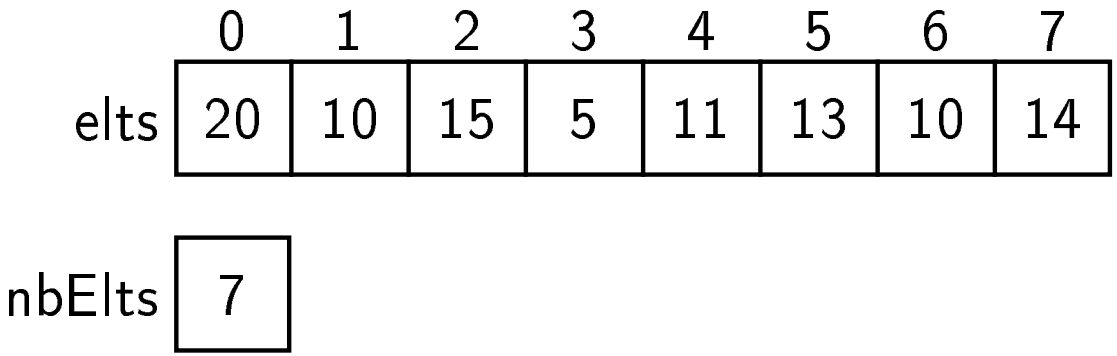


Le tableau elts est alors complètement rempli, il n'est plus possible de lui ajouter d'autes éléments.

Vous remarquerez que dans ce cas que **nbElts** est égal à **elts.length**

Tableau non rempli (4)

Aprés la suppression du dernier élément



Il est alors possible d'ajouter un élément 17 par exemple

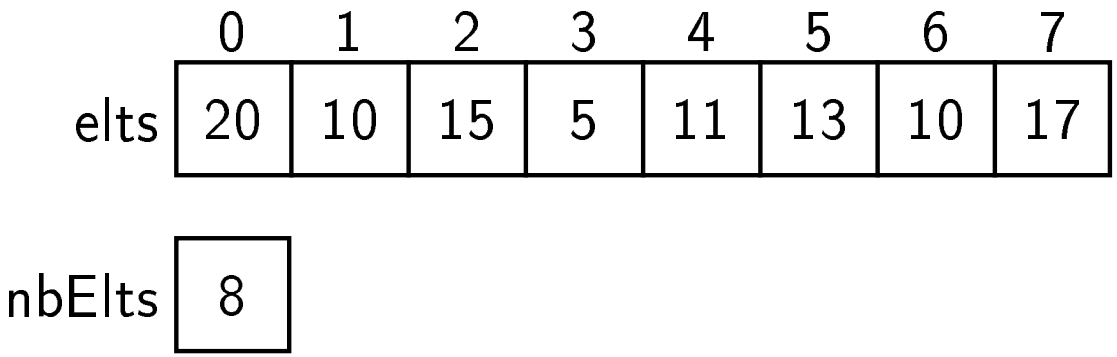


Tableau non rempli (5)

Aprés la suppression du premier élément (d'indice 0) on obtient

	0	1	2	3	4	5	6	7
elts	10	15	5	11	13	10	17	17

nbElts	7

On peut alors insérer un élément en position 0 , par exemple 50

	0	1	2	3	4	5	6	7
elts	50	10	15	5	11	13	10	17

nbElts	8

Tableau non rempli (6)

On peut aussi supprimer un élément d'indice donné, par exemple celui d'indice 3

	0	1	2	3	4	5	6	7
elts	50	15	10	15	11	13	17	17

nbElts	7

On peut également insérer un élément à une position donnée, par exemple 60 en position 0.

	0	1	2	3	4	5	6	7
elts	50	10	60	15	11	13	10	17

nbElts	8

Tableau non rempli (7)

Ecrire les fonctions dont les en-têtes sont :

```
public static void afficher ( int [ ] elts , int nbElts ) {
    // afficher les elements d'indice 0 a nbElts - 1 du tableau elts
}

public static int ajouter ( int [ ] elts , int nbElts , int eltAjout ) {
    // si la place est suffisante alors
    //    ajouter eltAjout a la fin du tableau elts
    // sinon
    //    afficher un message d'erreur
    // fsi
    // retournber nbElts
}

public static int supprimer ( int [ ] elts , int nbElts ) {
    // s'il existe un element a supprimer
    //    decrementer nbElts
    // sinon
    //     afficher un message d'erreur
    // fsi
    // retourner nbElts
}
```

Tableau non rempli (8)

Ecrire les fonctions dont les en-têtes sont :

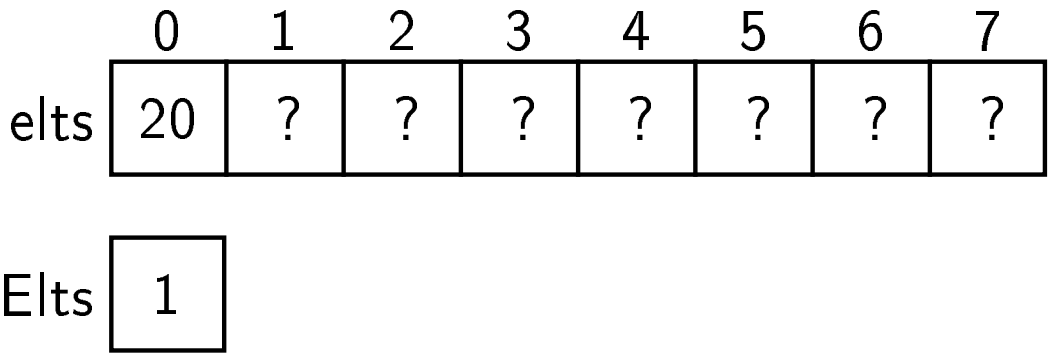
```
public static int inserer (
    int [ ] elts , int nbElts , int eltIns , int pos ) {
    // si l'insertion est possible
    //           et pos possede une valeur correcte alors
    //    inserer eltIns en postion pos
    // sinon
    //     afficher un message d'erreur
    // fsi
    // retourner nbElts
}

public static int supprimer ( int [ ] elts , int nbElts , int pos ) {
    // si la suppression est possible est possible
    //           et pos possede une valeur correcte alors
    //    supprimer l'element en position pos
    // sinon
    //     afficher un message d'erreur
    // fsi
    // retourner nbElts
}
```

Tableau non rempli (9)

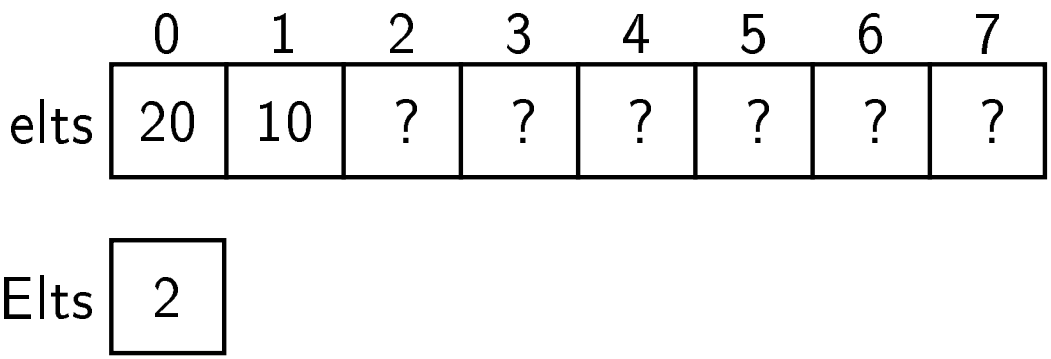
```
package tp7 ;

public class TableauNonRempli {


    public static void afficher ( int [ ] elts , int nbElts ) {

    }

    public static int ajouer ( int [ ] elts , int nbElts , int eltAjout ) {

            ...
            return nbElts ;
    }

    public static void main ( String [ ] args ) {
        int [ ] elts ;
        elts = new int [ 8 ] ;
        int nbElts ;
        nbElts = 0 ;
        elts = ajouter ( elts , nbElts , 20 ) ;
        afficher ( elts , nbElts ) ;

    }
}
```

Tableau non rempli (2)

Aprés l'ajout de 20



Aprés l'ajout de 10



Expressions conditionnelles

Une expression conditionnelle est une expression dont la valeur est vrai ou faux.

Elle utilise les opérateurs

- relationnels `==` `!=` `>` `>=` `<` `<=`
- logiques `!` `&&` `||`

Elle peut être

- affectée à une variable de type booléen
- ou être la condition d'une alternative ou d'une répétition.

L'opérateur non

p	! p
false	true
true	false

L'opérateur et non court-circuit

p	q	p & q
false	false	false
false	true	false
true	false	false
true	true	true

L'opérateur et court-circuit

p	q	p && q
false	?	false
true	false	false
	true	true

Si le premier opérande **p** est faux le deuxième opérande **q** n'est pas évalué (il peut être indéfini)

L'opérateur ou non court-circuit

p	q	p | q
false	false	false
false	true	true
true	false	true
true	true	true

L'opérateur ou court-circuit

p	q	p || q
true	?	true
false	false	false
	true	true

Si le premier opérande **p** est vrai le deuxième opérande **q** n'est pas évalué (il peut être indéfini)

Expressions conditionnelles Exemple 1

L'expression conditionnelle $x \in [a, b]$ s'écrit :

```
x >= a && x <= b
```

Expressions conditionnelles Exemple 2

L'expression conditionnelle n est pair s'écrit :

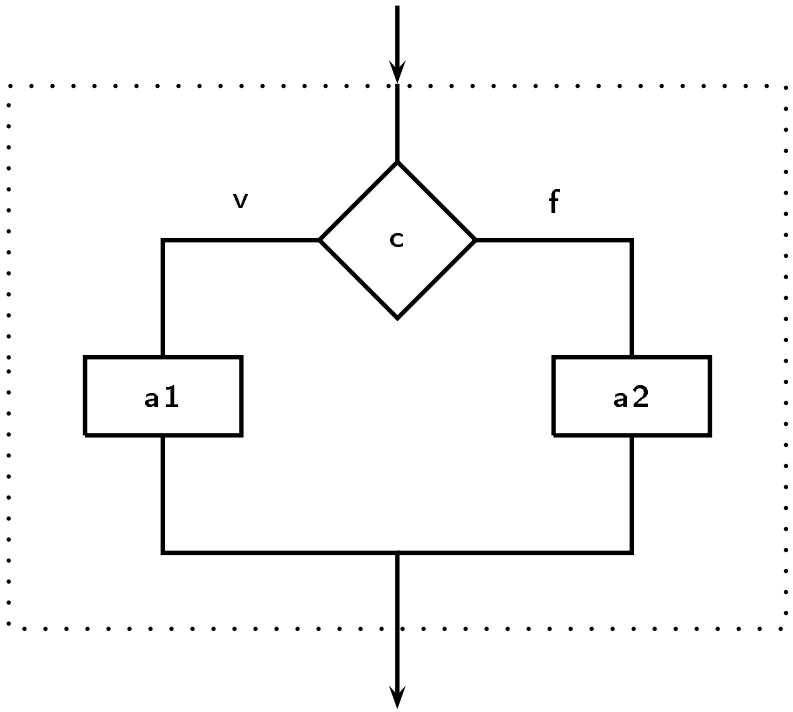
```
n % 2 == 0
```

Expressions conditionnelles Exemple 3

L'expression conditionnelle : la variable **car** de type **char** est un chiffre décimal s'écrit :

```
car >= '0' && car <= '9'
```

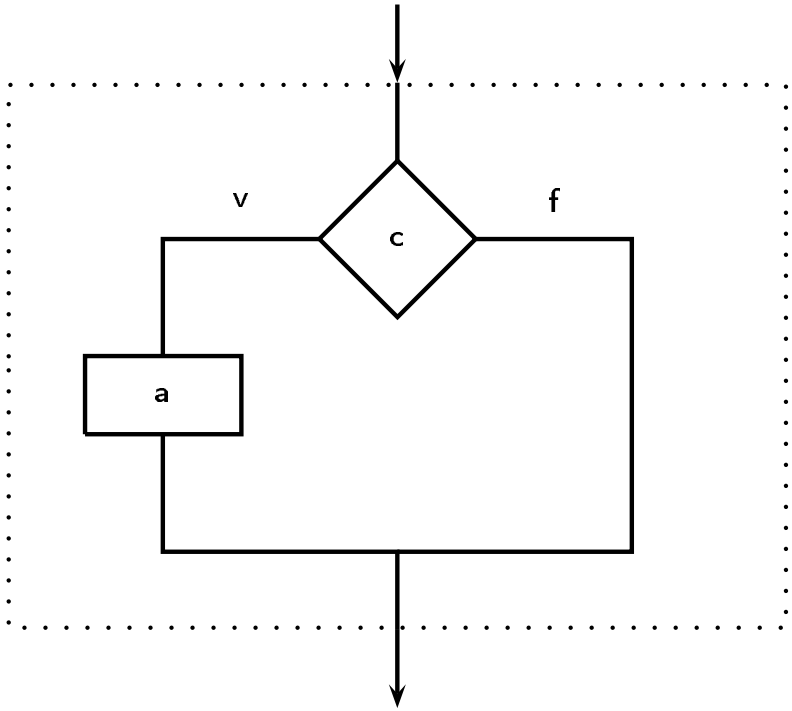
Alternative




```
if ( c ) {
    a1
} else {
    a2
}
```

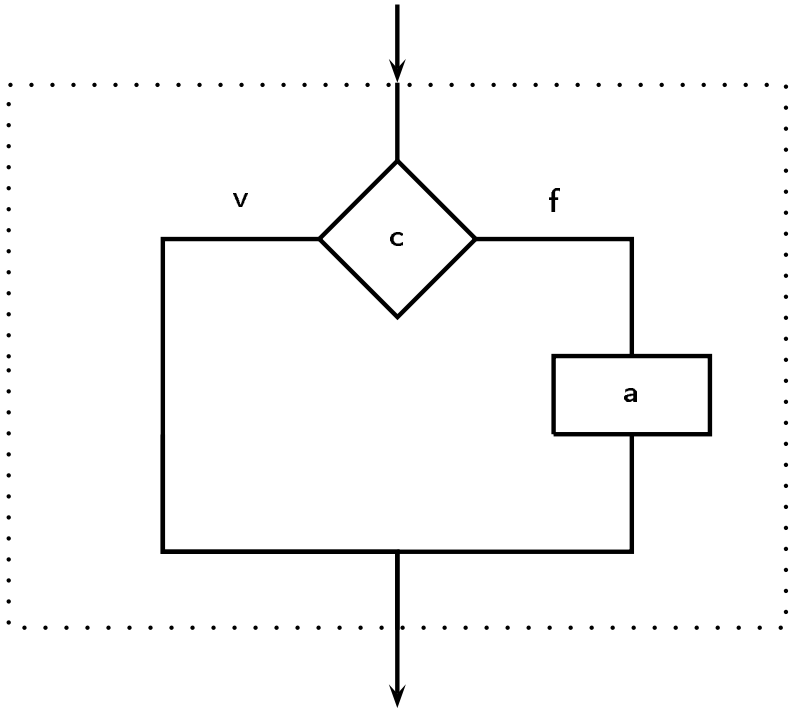
La condition **c** doit être une expression booléenne (dont la valeur est égale à vrai ou faux)
Si la condition est vraie, l'action **a1** est exécutée, dans le cas contraire l'action **a2** est exécutée

La partie sinon peut être vide




```
if ( c ) {
    a
}
```

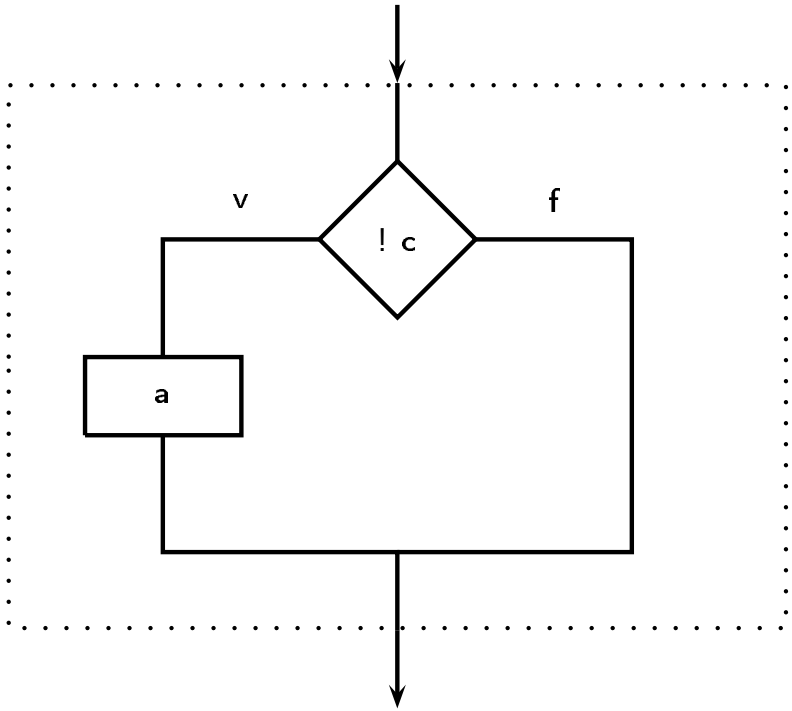
La partie alors peut être vide




```
if ( c ) {

} else {
    a
}
```

La partie alors peut être vide (suite)

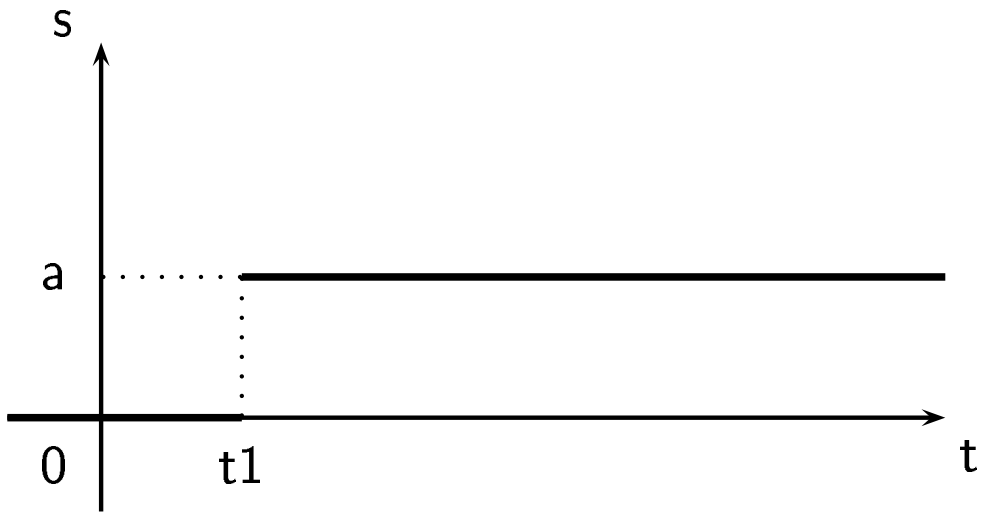
Dans ce cas il est préférable de transformer l'alternative sous la forme suivante :




```
if ( ! c ) {
    a
}
```

Alternative Exemple 1

Donner à la variable **s** la valeur de la fonction définie par le diagramme :



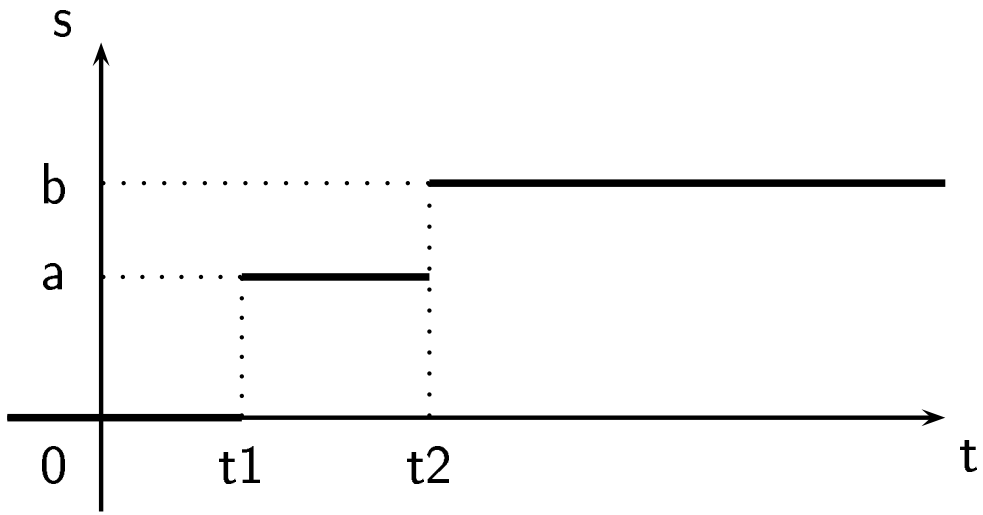
On supposera que les varaiables **a**, **t1** et **t** ont été déclarées et qu'elles ont reçu une valeur.

Alternative Exemple 1

```
if ( t < t1 ) {
     s = 0 ;
} else {
     s = a ;
}
```

Alternative Exemple 2

Donner à la variable **s** la valeur de la fonction définie par le diagramme :



On supposera que les varaiables **a**, **b**, **t1**, **t2** et **t** ont été déclarées et qu'elles ont reçu une valeur.

Alternative Exemple

```
if ( t < t1 ) {
     s = 0 ;
} else if ( t < t2 {
     s = a ;
} else {
     s = b ;
}
```

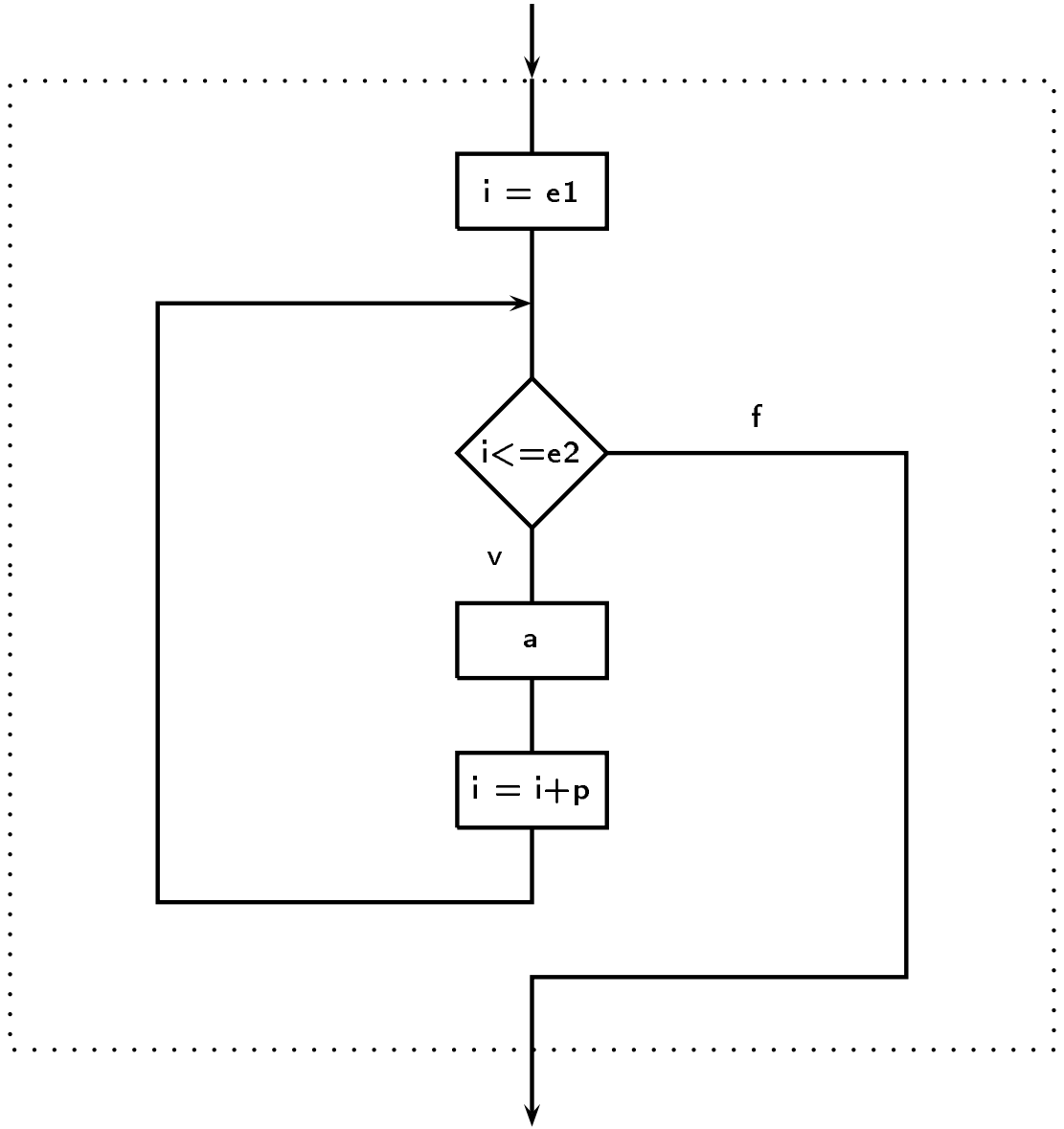
Répétitions

Répétitions dont le nombre est connu la boucle **pour**

La boucle pour avec un pas positif p

```
pour i de e1 à e2 pas p faire
    a
finpour
```

Répétition pour avec un pas positif p



Répétition pour avec un pas positif p

La variable **i** dite variable de contrôle

- est initialisée avec la valeur de **e1**
- si elle est inférieure ou égale à celle de **e2**
  - l'action **a** est exécutée
  - on ajoute ensuite le pas **p** à **i**
  - puis enfin on retourne à la comparaison avec **e2**
- si elle est supérieure à celle de **e2** on sort de la boucle pour

Remarque

- si la valeur de **e1** est supérieure à celle de **e2** l'action **a** n'est pas exécutée.

Répétition pour avec un pas p positif

```
for ( int i = e1 ; i <= e2 ; i = i + p) {

    a

}
```

Répétition pour avec un pas p positif Exemple 1

Donner les instructions qui permettent de calculer la somme des carrés des nombres pairs compris entre 2 et 100.

$$s2 = 2^2 + 4^2 + \ldots + 98^2 + 100^2$$

Répétition pour avec un pas p positif Exemple 1

```
int s2;
s2 = 0 ,
for ( int i = 2 ; i <= 100 ; i = i + 2) {

    s2 = s2 + i * i ;

}
```

Répétition pour avec un pas p positif Exemple 2

Donner les instructions qui permettent d'afficher **n1** lignes de **n**2 étoiles.
Par exemple pour **n1**=3 et **n2**=8 on doit obtenir :

```
********
********
********
```

On supposera que les variables entières **n1** et **n2** ont été déclarées et ont reçu une valeur correcte.

Répétition pour avec un pas p positif Exemple 2

```
for ( int i = 1 ; i <= n1 ; i = i + 1) {

    // afficher n2 etoiles en restant sur la ligne

    // passer a la ligne

}
```

Répétition pour avec un pas p positif Exemple 2

```
for ( int i = 1 ; i <= n1 ; i = i + 1) {
    for ( int j = 1 ; j <= n2 ; j = j + 1 ) {
        System.out.print ( "*" ) ;
    }
    System.out.println ( ) ;
}
```

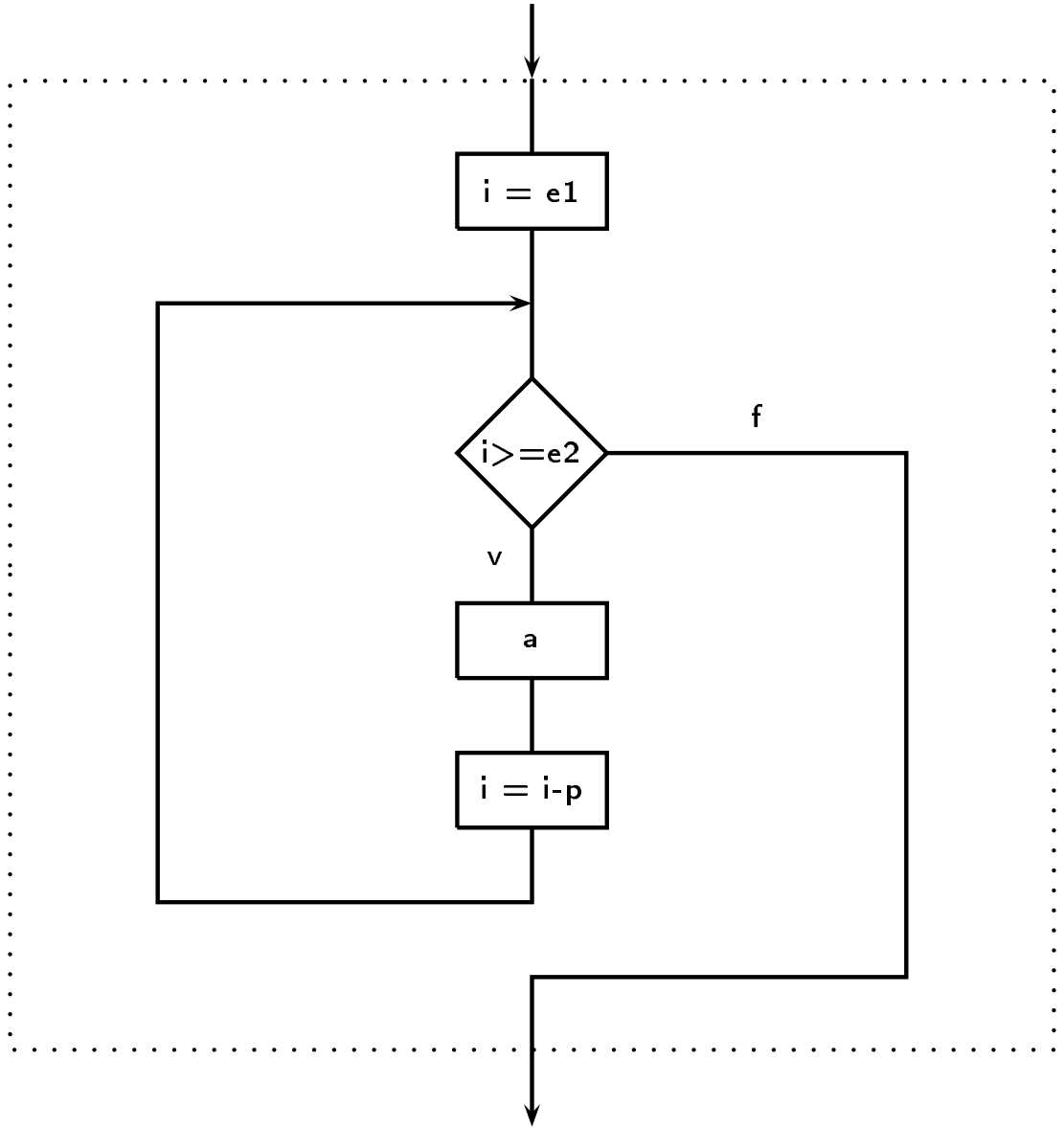
Répétitions

Répétitions dont le nombre est connu la boucle **pour**

La boucle pour avec un pas négatif -p

```
pour i de e1 à e2 pas -p faire
    a
finpour
```

Répétition pour avec un pas négatif -p



Répétition pour avec un pas négatif -p

La variable **i** dite variable de contrôle

- est initialisée avec la valeur de **e1**
- si elle est supérieure ou égale à celle de **e2**
  - l'action **a** est exécutée
  - on soustrait ensuite le pas **p** à **i**
  - puis enfin on retourne à la comparaison avec **e2**
- si elle est inférieure à celle de **e2** on sort de la boucle pour

Remarque

- si la valeur de **e1** est inférieure à celle de **e2** l'action **a** n'est pas exécutée.

Répétition pour avec un pas p négatif

```
for ( int i = e1 ; i >= e2 ; i = i - p) {

    a

}
```

Répétition pour Exercice 1

Quel est le résultat de l'exécution des instructions suivantes :

```
int e1 ;
e1 = 1 ;
int e2 ;
e2 = 6 ;
System.out.println ( "Debut" ) ;
for ( int i = e1 ; i <= e2 ; i = i + 2 ) {
    System.out.print ( i + "␣" ) ;
}
System.out.println ( ) ;
System.out.println ( "Fin" ) ;
```

Répétition pour Exercice 2

Quel est le résultat de l'exécution des instructions suivantes :

```
int e1 ;
e1 = 6 ;
int e2 ;
e2 = 1 ;
System.out.println ( "Debut" ) ;
for ( int i = e1 ; i <= e2 ; i = i + 2 ) {
    System.out.print ( i + "␣" ) ;
}
System.out.println ( ) ;
System.out.println ( "Fin" ) ;
```

Répétition pour Exercice 3

Quel est le résultat de l'exécution des instructions suivantes :

```
int e1 ;
e1 = 1 ;
int e2 ;
e2 = 6 ;
System.out.println ( "Debut" ) ;
for ( int i = e1 ; i >= e2 ; i = i - 2 ) {
    System.out.print ( i + "␣" ) ;
}
System.out.println ( ) ;
System.out.println ( "Fin" ) ;
```

Répétition pour Exercice 4

Quel est le résultat de l'exécution des instructions suivantes :

```
int e1 ;
e1 = 6 ;
int e2 ;
e2 = 1 ;
System.out.println ( "Debut" ) ;
for ( int i = e1 ; i >= e2 ; i = i - 2 ) {
    System.out.print ( i + "␣" ) ;
}
System.out.println ( ) ;
System.out.println ( "Fin" ) ;
```

Tableaux

Un tableau doit être

- déclaré
- puis construit ou initialisé

Les tableaux déclaration

La déclaration d'un tableau est de la forme

```
typeEltsTab [  ] nomTab  ;
```

- **typeEltsTab** est le type des éléments du tableau.
- **nomTab** est le nom du tableau.
- les crochets indiquent que l'on déclare un tableau.

  Si on les oublie on déclare une variable simple.

Les tableaux construction

La construction d'un tableau est de la forme :

```
nomTab = new typeEltsTab [ nbEltsTab ] ;
```

- **nomTab** est le nom du tableau.
- **typeEltsTab** est le type des éléments du tableau.
- **nbEltsTab** est le nombre d'éléments du tableau.

  C'est une expression dont la valeur est définie ≥ 0.

Tableaux initialisation

L'initialisation qui permet de construire un tableau en donnant des valeurs à ses éléments est de la forme :

```
nomTab = new typeEltsTab [ ] { val1 , ... , valn }
```

Tableaux nombre d'éléments et indices

- Le nombre d'éléments du tableau **nomTab** est **nomTab.length**
- Les éléments du tableau **nomTab** sont indicés
  - de **0**
  - à **nomTab.length - 1**
- L'accès à un élément du tableau **nomTab** déclenche une exception **ArrayIndexOutOfBoundsException** si l'indice n'est pas compris entre **0** et **nomTab.length - 1**

Tableaux Exercice 1

Quels sont les résultats affichés par l'exécution des instructions suivantes :

```
double [ ] t ;
t = new double [ ] { 20.0 , 10.0 , 30.0 , 40.0 } ;
int k ;
k = 1 ;
for ( int i = 0 ; i <= t . length - 1 ; i = i + 1 ) {
    t [ i ] = t [ i ] / t [ k  ] ;
}
Tableaux.afficher ( t ) ;
```

Tableaux Exercice 2

Quels sont les résultats affichés par l'exécution des instructions suivantes :

```
int n ;
n = 5 ;
int [ ] c ;
c = new int [ n + 1 ] ;
for ( int i = 0 ; i <= n ; i = i + 1 ) {
    c [ i ] = 1 ;
    for ( int j = i - 1 ; j >= 1 ; j = j - 1 ) {
        c [ j ] = c [ j ] + c [ j - 1 ] ;
    }
    for ( int j = 0 ; j <= i ; j = j + 1 ) {
        System.out.print ( c [ j ] + "␣" ) ;
    }
    System.out.println ( ) ;
}
```

Déclaration ou définition d'une fonction

La déclaration d'une fonction (avant la fonction main) est de la forme :

```
public static typeValRet nomFct ( listParamForm ) {

      // corps de la fonction

}
```

- **typeValRet** est le type de la valeur retournée.
  - **void** indique que la fonction ne retourne rien
- le nom de la fonction **nomFct** doit indiquer clairement le traitement effectuée par la fonction.
- **listParamForm** est la liste des paramètres formels

Appel d'une fonction

L'appel d'une fonction (dans le main) est de la forme :

- dans le cas où la fonction est définie dans la classe où elle utilisée.

```
nomFct ( listeParamEff ) ;
```

- dans le cas contraire :

```
NomClasse.nomFct ( listeParamEff ) ;
```

  - **listParamEff** est la liste des paramètres effectifs.
  - **NomClasse** est le nom de la classe où est définie la fonction.

Les parenthèses sont obligatoires même si la liste des paramètres effectifs est vide.

Fonctions Exercice 1

Ecrire la fonction **estUnChiffreDec** qui retourne **true** si la caractère **car** passé en paramètre est un chiffre décimal et **false** dans le cas contraire.

Fonctions Exercice 1

```
public static boolean estUnChiffreDec ( char car ) {
    boolean result ;
    result = car >= '0' && car <= '9' ;
    return result ;
}
```

ou

```
public static boolean estUnChiffreDec ( char car ) {
    return car >= '0' && car <= '9' ;
}
```

Fonctions Exercice 2

Ecrire la fonction **min2** qui retourne le minimum de ses deux paramètres entiers **a** et **b**.

Fonctions Exercice 2

```
public static int min2 ( int a , int b ) {
    int min ;
    if ( a < b ) {
        min = a ;
    } else {
        min = b ;
    }
    return min ;
}
```

Fonctions Exercice 2

```
public static int min2 ( int a , int b ) {
    if ( a < b ) {
        return a ;
    } else {
        return b ;
    }
}
```

Fonctions Exercice 3

Ecrire la fonction **min3** qui retourne qui retourne le minimum de ses deux paramètres entiers **a**, et **b** et **c**.

Fonctions Exercice 3

```
public static int min3 ( int a , int b , int c ) {
    int min ;
    min = min2 ( a , b ) ;
    if ( c < min ) {
        min = c ;
    }
    return min ;
}
```

Fonctions Exercice 3

```
public static int min3 ( int a , int b , int c ) {
    return min2 ( min2 ( a , b ) , c ) ;
}
```

Fonctions Exercice 4

Ecrire la fonction **minTab** qui retourne qui retourne le minimum du tableau d'entiers **tab** passé en paramètre.

Fonctions Exercice 4

```
public static int minTab ( int [ ] tab ) {
    int min ;
    min = tab [ 0 ] ;
    for ( int i = 1; i <= tab.length - 1; i = i - 1 )
        if ( tab [ i ] < min ) {
            min = tab [ i ] ;
        }
    }
    return min ;
}
```

Tableau non rempli (1)

Le tableau **elts** contient 0 éléments, il pourra en contenir **elts.length**

nbElts donne le nombre d'éléments du tableau **elts**

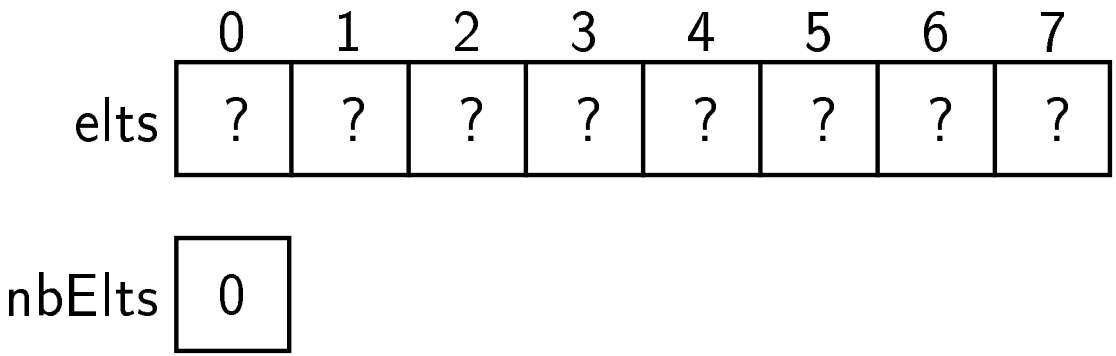
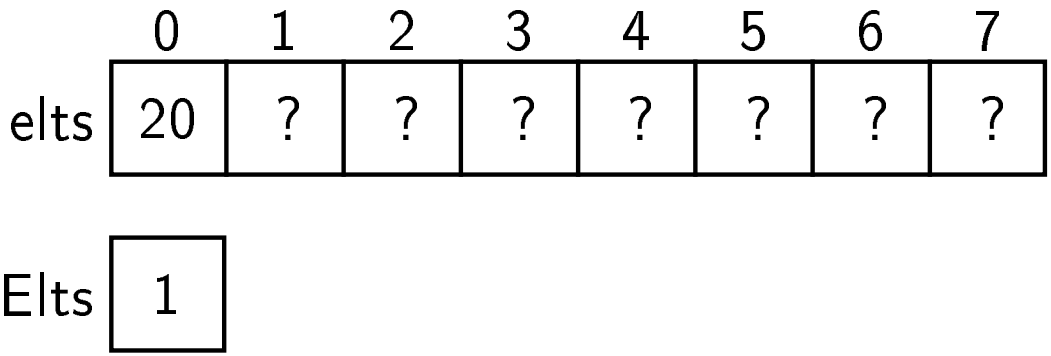


Tableau non rempli (2)

Aprés l'ajout de 20



Aprés l'ajout de 10

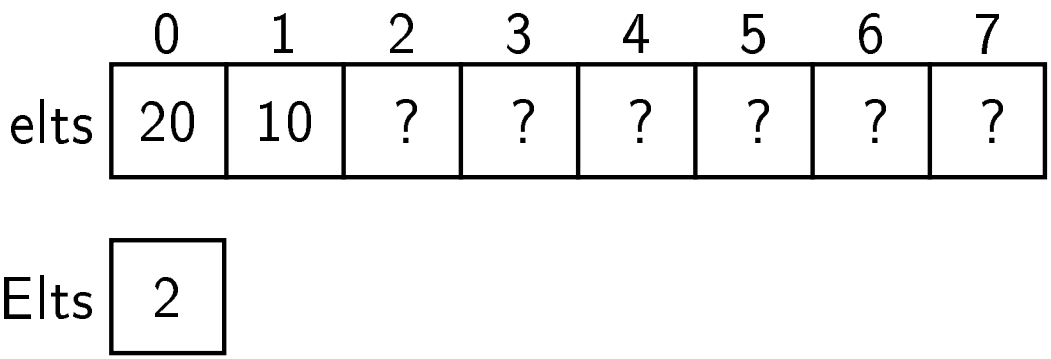
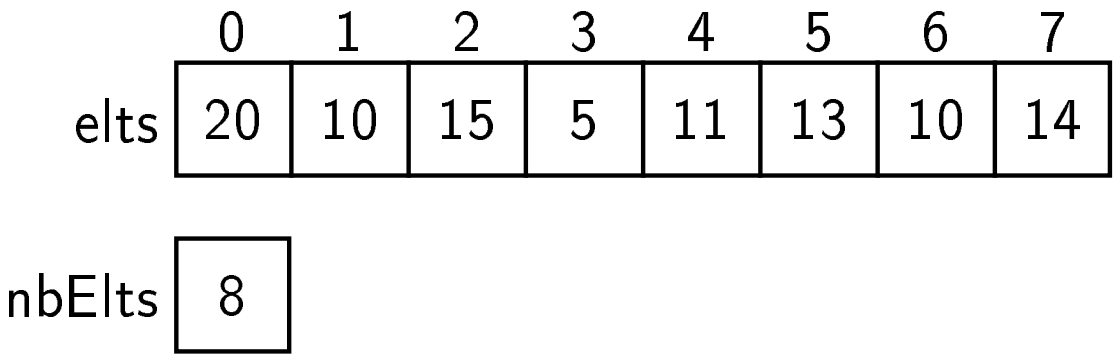


Tableau non rempli (3)

Aprés l'ajout de 15, 5, 11, 13, 10 et 14



Le tableau **elts** est alors complètement rempli, il est impossible de lui ajouter d'autes éléments.

Vous remarquerez que dans ce cas que **nbElts** est égal à **elts.length**

Tableau non rempli (4)

Ecrire la fonction **ajouter** qui permet d'ajouter un élément au tableau **elts**.

Tableau non rempli (5)

```
public static int ajouter ( int [ ] elts , int nbElts , int eltAjout ) {
    /*****************
    si la place est suffisante alors
        ajouter eltAjout a la fin du tableau elts
    sinon
        afficher un message d'erreur
    fsi
    retourner nbElts
    *****************/
}
```

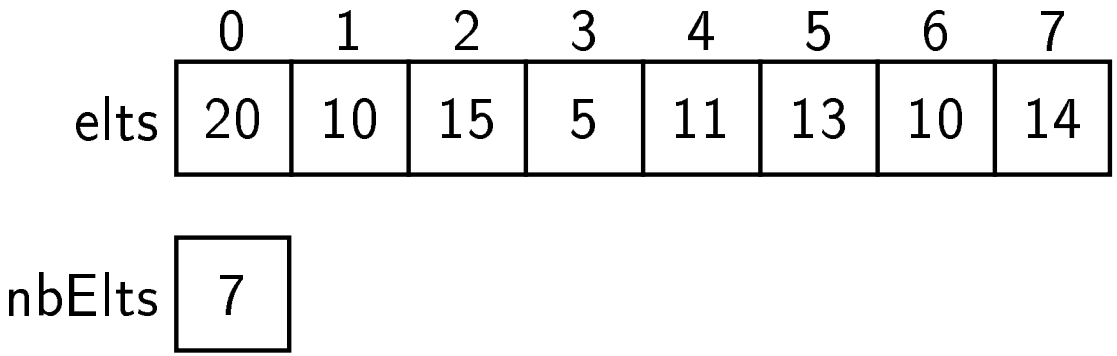
Pourquoi faut-il retourner **nbElts** ?

Tableau non rempli (6)

```
public static int ajouter ( int [ ] elts , int nbElts , int eltAjout ) {
    if ( nbElts < elts.length ) {
        elts [ nbElts ] = eltAjout ;
        nbElts = nbElts + 1 ;
    } else {
        System.out.println ( "ERREUR: ajout impossible tableau plein" ) ;
    }
    return nbElts ;
}
```

Tableau non rempli (7)

Aprés la suppression du dernier élément



Il est alors possible d'ajouter un élément 17 par exemple

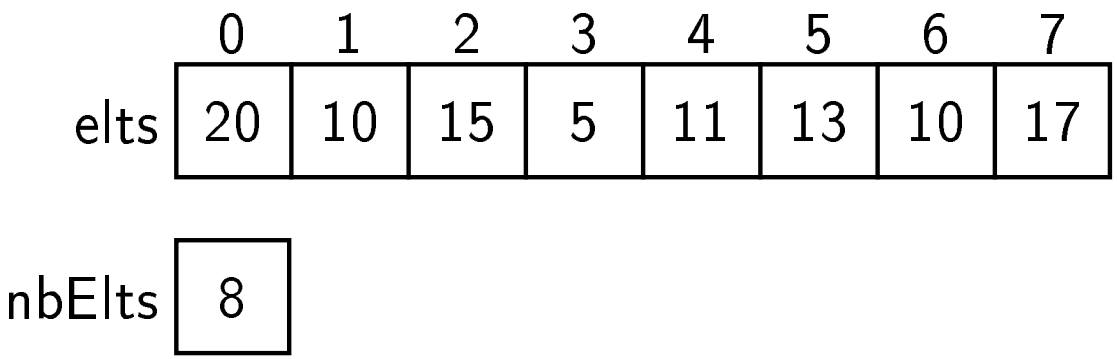


Tableau non rempli (8)

Ecrire la fonction **supprimer** qui permet de supprimer le dernier élément du tableau **elts**.

Tableau non rempli (9)

```
public static int supprimer ( int [ ] elts , int nbElts ) {
     /*************************
     s'il existe un element a supprimer
          decrementer nbElts
     sinon
          afficher un message d'erreur
     fsi
     retourner nbElts
     ************************ */
}
```

Tableau non rempli (9)

```
public static int supprimer ( int [ ] elts , int nbElts ) {
    if ( nbElts != 0 ) {
        nbElts = nbElts - 1 ;
    } else {
        System.out.println ( "ERREUR: suppression impossible tableau vide" ) ;
    }
    return nbElts ;
}
```

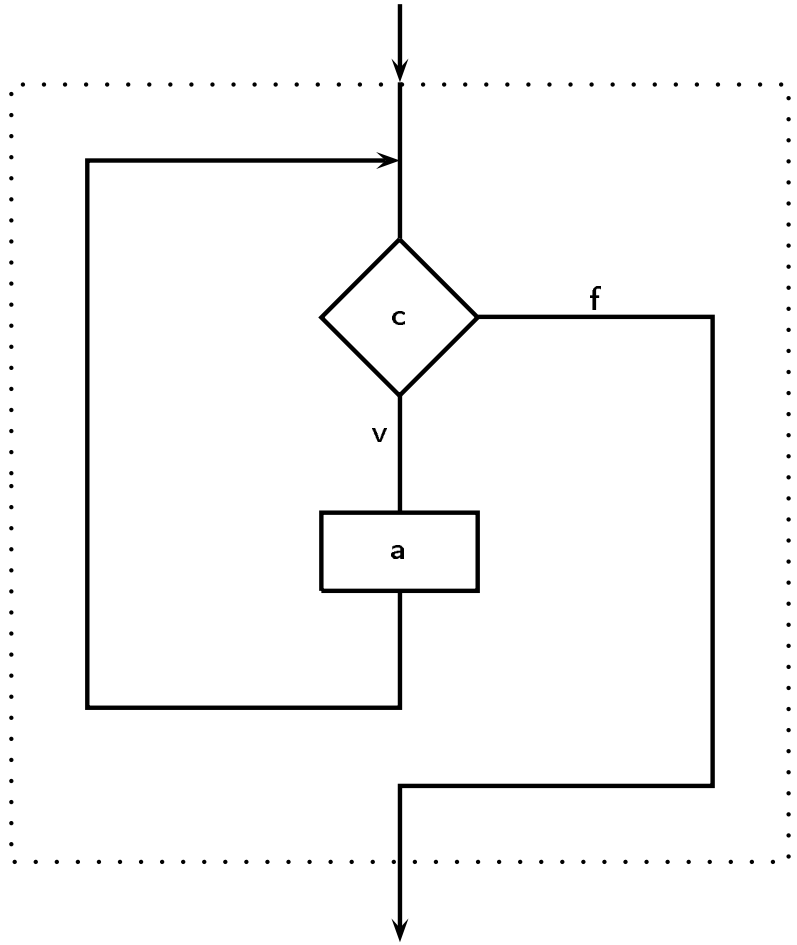
Répétitions dont le nombre n'est pas connu : la boucle tantque faire

```
tq  c  faire {
    a
ftq
```

```
while ( c ) {
    a
}
```

Répétitions dont le nombre n'est pas connu : la boucle tantque faire

```
while ( c ) {

    a

}
```



Répétitions dont le nombre n'est pas connu : la boucle tantque faire

- On commence par évaluer la condition **c**
  - si elle est vérifiée, l'action **a** est exécutée, puis on teste à nouveau la condition **c**.
  - si elle n'est pas vérifiée, l'action **a** n'est pas exécutée (on sort du **tantque**)
- La condition **c** doit être définie à l'entrée du **tantque**
- L'action **a** doit modifier la condition **c** pour permettre une sortie du **tantque**.

La boucle tantque faire : Exemple

Calcul de la somme d'une suite de nombres positifs ou nuls saisis à partir du clavier, le dernier nombre est suivi de -1

Par exemple : 10,9,11 et -1.

La boucle tantque faire : Exemple

```
public static int somme ( ) {


}
```

La boucle tantque faire : Exemple

```
public static int somme ( ) {
    int som ;
    som = 0 ;
    int nbre ;
    nbre = EntreeClavier.readInt("nbre␣?␣") ;
    while ( nbre != -1 ) {
        som = som + nbre ;
        nbre = EntreeClavier.readInt("nbre␣?␣") ;
    }
    return som ;
}
```

La boucle tantque faire : Exemple

Ecrire la fonction **saisieNote** qui permet de saisir à partir du clavier un nombre entier compris entre 0 et 20.

Tant que l'utilisateur ne saisit pas un nombre compris entre 0 et 20, un message d'erreur est affiché et on repropose une nouvelle saisie.

Cette fonction n'admet pas de paramètre et retourne la note saisie obligatoirement comprise entre 0 et 20

La boucle tantque faire : Exemple

```
public static int saisirNote ( ) {


}
```

La boucle tantque faire : Exemple

```
public static int saisirNote ( ) {
    int n ;
    n = EntreeClavier.readInt( "note ? " );
    while ( n < 0 || n > 20 ) {
        System.out.print ( "Erreur" ) ;
        System.out.println ( "n doit etre compris entre 0 et 20" ) ;
        n = EntreeClavier.readInt( "note ? " );
    }
    return n ;
}
```

Recherche séquentielle d'une valeur dans un tableau non rempli

Ecrire la fonction **appartient** qui admet trois paramètres

- **a** un tableau d'entiers dans lequel s'effectue la recherche
- **nbElts** entier égal au nombre d'éléments du tableau précédent
- **valRech** entier valeur recherchée dans le tableau

et qui retourne **vrai** si la valeur recherchée **valRech** appartient au tableau **a** et **faux** dans le cas contaire.

La fonction appartient

```
public static boolean appartient ( int [ ] a , int nbElts , int valRech ) {


}
```

La fonction appartient

```
public static boolean appartient (  int [ ] a , int nbElts , int valRech) {
    boolean trouve ;
    trouve = false ;
    int i ;
    i = 0 ;
    while ( i < nbElts && ! trouve ) {
        if ( a [ i ] == valRech ) {
            trouve = true ;
        } else {
            i = i + 1 ;
        }
    }
    return trouve ;
}
```

Elimination des doublons

Soit le tableau non rempli :

	0	1	2	3	4	5	6	7	8	9	10	11	12
a	10	21	30	30	30	70	21	10	50	50	40	?	?

na	11

Comment obtenir :

	0	1	2	3	4	5	6	7	8	9	10
b	10	21	30	70	50	40	?	?	?	?	?

nb	6

Fonction elimDoublons

```
public static int elimDoublons (  int [ ] a , int na , int [ ] b ) {

    // le tableau b doit etre construit avant l'appel de cette fonction

    // cette fonction retourne nb
}
```

Fonction elimDoublons

```
public static int elimDoublons (  int [ ] a , int na , int [ ] b ) {
    int nb ;
    nb = 0 ;
    for ( int i = 0 ; i < na ; i = i + 1 ) {
        if ( ! appartient ( b , nb , a [ i ] ) {
            b [ nb ] = a [ i ] ;
            nb = nb + 1 ;
        }
    }
    return nb ;
}
```

```
public static void main ( String [ ] args ) {
    int [ ] a ;
    a = new int [ ] { 10,21,30,30,30,70,21,10,50,50,40,0,0} ;
    int na ;
    na = 11 ;
    int [ ] b ;
    b = new int [ na ] ;
    int nb ;
    nb = elimDoublons ( a , na , b ) ;
    for ( int i = 0 ; i < nb ; i = i + 1 ) {
        System.out.print ( b [ i ] + "␣" ) ;
    }
    System.out.println ( ) ;
}
```

L'instruction **for** du langage Java n'est qu'une forme abrégée de l'instruction **while**

```
for ( init ; cond ; incr ) {
    a
}
```

équivaut à

```
init ;
while ( cond ) {
    a
    incr ;
}
```

L'instruction **for** avec un pas **p** positif

```
for ( int i = e1 ; i <= e2 ; i = i + p) {

    a

}
```

équivaut à

```
int i = e1;

while ( i <= e2 ) {
    a

    i = i + p ;
}
```

L'instruction **for** avec un pas -**p** négatif :

```
for ( int i = e1 ; i >= e2 ; i = i - p) {

    a

}
```

équivaut à

```
int i = e1;

while ( i >= e2 ) {
    a

    i = i - p ;
}
```

Les caractères et les chaînes de caractères

Les types

- char
- String

Le type char

1. Les littéraux caractères
2. Déclaration d'une variable de type caractère
3. Affectation
4. Saisie à partir du clavier
5. Affichage à l'écran
6. Code Unicode d'un caractère
7. Transformer un code Unicode en caractère
8. Comparaison
9. Transformations majuscules ou minuscules

Les littéraux caractères

Les valeurs des caractères sont représentés par des littéraux

Les littéraux caractères sont encadrés par des aspostrophes et non par des guillemets

Des caractères visibles ou imprimables

La lettre A majuscule	'A'
La lettre a minuscule	'a'
Le chiffre 1	'1'
L'espace	' '
La virgule	','

Des caractères de contrôle

La fin de ligne (new line)	'\n'
Le retour chariot (carriage return)	'\r'
La tabulation	'\t'
Le caractère apostrophe	'\''

Des caractères en Unicode (généralisation du code Ascii sur 16 bits)

La lettre A majuscule	'\u0041'
La lettre a miniscule	'\u0061'
Le chiffre 1	'\u0031'
La fin de ligne (new line)	'\u000a'
Le retour chariot (carriage return)	'\u000d'
La tabulation	'\u0009'

Déclaration et affectation

Une variable **c** de type caractère doit être déclarée

```
char c ;
```

On peut alors lui affecter un littéral caractère

```
c = 'A' ;
```

ou la valeur d'une autre variable de type caractère

```
char c1 ;
  ...
c = c1 ;
```

Saisie et affichage

Saisie à partir du clavier

```
c = EntreeClavier.readChar ( "? " ) ;
```

Affichage à l'écran en restant sur la ligne

```
System.out.print ( c ) ;
```

Affichage à l'écran en passant à la ligne

```
System.out.println ( c ) ;
```

Code Unicode d'un caractère

Pour obtenir le code unicode d'une variable c de type caractère, il suffit de ranger sa valeur dans une variable entière.

Les instructions suivantes permttent d'afficher le code Ascii de la lettre **A** majuscule.

```
char c ;
c = 'A' ;
int code ;
code = c ;
System.out.println ( code ) ;
```

Obtenir un caractère à partir de son code Unicode

```
int code ;
code = 65 ;
char c ;
c = ( char ) code ;
System.out.println ( c ) ;
```

Le cast (char) permet de transformer la valeur entière de code de 32 bits(4 octets) en une valeur caractère de 16 bits (2 octets).

Comparaison

Les caractères sont considérés comme de valeurs numériques, il est donc possible d'utiliser les opérateurs de comparaison.

L'expression :

```
c == 'Y' || c == 'y'
```

prend la valeur vrai si c est égal à lettre Y majuscule ou minuscule.

L'expression :

```
c >= '0' && c <= '9'
```

prend la valeur vrai si c est un chiffre décimal.

Transformations en majuscules ou en minuscules

La fonction **Character.toUpperCase** retourne la caractère passé en paramètre en majuscule.

Par exemple

```
char c ;
c = 'a' ;
c = Character . toUpperCase ( c ) ;
System.out.println ( c ) ;
```

La fonction **Character.toLowerCase** retourne la caractère passé en paramètre en minuscule.

Exercices

- La fonction estChiffreHexa
- Table Ascii en décimal
- Table Ascii en hexadécimal

Le type String

1. Les littéraux chaînes de caractères
2. Déclaration d'une variable de type chaîne de caractères
3. Affectation
4. Saisie à partir du clavier et affichage à l'écran
5. Longueur d'une chaîne de caractères
6. Accès au caractère de rang i
7. Concaténation
8. Egalité de deux chaînes de caractères
9. Ordre lexicographique
10. Comparaison de deux chaînes de caractères
11. Recherche d'un caractère
12. Conversion d'une chaîne en nombres
13. Conversion d'un nombre en chaîne
14. Extraction d'une sous-chaîne

Les littéraux chaînes de caractères

Les valeurs des chaînes caractères sont représentés par des littéraux

Les littéraux chaînes de caractères sont encadrés par des guillemets et non par des aspostrophes

"RT-1"	4 caractères
""	chaîne vide aucun caractère
"A"	un seul caractère
" "	un espace
"abc\n12"	6 caractères
"\""	un guillemet

Il est très important de ne pas confondre le caractère `'A'` et la chaîne de caractères `"A"`

Déclaration et affectation

Une variable **ch** de type chaîne de caractères doit être déclarée

```
String ch ;
```

On peut alors lui affecter un littéral chaîne de caractères

```
ch = "RT-1" ;
```

ou la valeur d'une autre variable de type chaîne de caractères

```
String ch1 ;
   ...
ch = ch1 ;
```

Saisie et affichage

Saisie à partir du clavier

```
ch = EntreeClavier.readString ( "? " ) ;
```

Affichage à l'écran en restant sur la ligne

```
System.out.print ( ch ) ;
```

Affichage à l'écran en passant à la ligne

```
System.out.println ( ch ) ;
```

Longueur d'une chaîne de caractères

L'expresssion

```
ch . length ( )
```

donne le nombre de caractères (ou longueur) de la chaîne de caractères ch

```
String ch1 ;
ch1 = "RT-1" ;
System.out.println ( ch1 . length ( ) ) ;
String ch2 ;
ch2 = "" ;
System.out.println ( ch2 . length ( ) ) ;
```

Longueur d'une chaîne de caractères (suite)

Ne pas confondre

- la longueur d'une chaîne de caractères
- et le nombre d'éléments d'un tableau

La longueur de la chaîne de caractères **ch** est

```
ch . length ( )
```

Le nombre d'éléments du tableau **a** est

```
a . length
```

Accès au caractère de rang i

L'expresssion

```
ch . charAt ( i )
```

donne le caractère de rang **i** de la chaîne **ch**

Le rang **i** doit être compris entre 0 et `ch . length ( ) -1`.

Dans le cas contraire une exception

```
StringIndexOutOfBoundsException
```

est déclenchée et l'exécution du programme est stoppée.

Remarque : `setCharAt` n'existe pas pour les chaînes de caractères.

Exercice

Ecrire la fonction qui permet d'afficher la chaîne de caractères passée en paramètre à l'envers.

```
public static void affiche Envers ( String ch ) {

            ...

}
```

Exercice

Ecrire la fonction qui retourne la chaîne de caractères passée en paramètre à l'envers.

```
public static String envers ( String ch ) {

            ...

}
```

Exercice

Ecrire la fonction qui retourne le nombre d'espaces de la chaîne de caractères passée en paramètre.

```
public static int nbEspaces ( String ch ) {

            ...

}
```

Concaténation

L'opérateur + permet de concaténer deux chaînes de caractères.

Par exemple

```
    ...
ch1 = "RT" ;
ch2 = "Béthune" ;
ch1 = ch1 + " " + ch2 ;
  // la valeur de ch1 est "RT Béthune"
```

Egalité de deux chaînes de caractères

Il n'est pas possible d'utiliser les opérateurs d'égalité ou d'inégalité avec les chaînes de caractères
L'expression

```
s1 . equals ( s2 )
```

retourne vrai si les deux chaînes sont égales et faux dans le cas contraire.

Egalité de deux chaînes de caractères (suite)

```
    ...
s1 = "ABC" ;
s2 = "abc" ;
System.out.println( s1.equals ( s2 ) ) ;
  // -> false
System.out.println( s1.equalsIgnoreCase ( s2 ) ) ;
  // -> true
```

Ordre lexicographique

La chaîne **s1** précède la chaîne **s2** (**s1** $\prec$ **s2**) **ssi**

- **s1** est un préfixe de **s2**

  bal $\prec$ ballon

- Le premier caractère de **s1** qui ne coïncide pas avec le caractère de même range de **s2** est inférieur à ce dernier

  ballon $\prec$ bas

Comparaison de deux chaînes de caractères

L'expression

```
s1 . compareTo ( s2 )
```

$$\text{retourne une valeur} \begin{cases} < 0 & \text{si} \quad s1 \prec s2 \\ 0 & \text{si} \quad s1 \text{ est égal à } s2 \\ > 0 & \text{si} \quad s1 \succ s2 \end{cases}$$

L'expression

```
s1 . compareToIgnoreCase ( s2 )
```

effectue la même comparaison sans tenir compte des majuscules et minuscules.

Recherche d'un caractère

L'expression

```
s . indexOf ( c )
```

retourne

- la première occurrence du caractère **c** dans la chaîne **s**
- -1 si le caractère **c** n'a pas été trouvé

Recherche d'un caractère

L'expression

```
s . indexOf ( c , i0 )
```

retourne

- la première occurrence du caractère **c** dans la chaîne **s** à partir du rang **i0**
- -1 si le caractère **c** n'a pas été trouvé

Exercice

Réécrire la fonction **nbEspaces** en utilisant la méthode **indexOf**.

```
public static int nbEspaces ( String ch ) {

            ...

}
```

Conversion d'une chaîne de caractères en nombre

L'expression

```
Integer . parseInt ( s )
```

donne la valeur numérique de la chaîne **s**

```
...
s = "125" ;
int v ;
v = Integer . parseInt ( s )
v = 2 * v ;
// la valeur de v est 250
```

Conversion d'une chaîne de caractères en nombre (suite)

Si la chaîne **s** ne représente pas un nombre entier une exception

```
NumberFormatException
```

est déclenchée et l'exécution du programme est stoppée.

Conversion d'un nombre en chaîne de caractères

L'expression

```
String . valueOf ( n )
```

donne la chaîne de caractères représentant le nombre n

```
String s ;
s = 125 ;
int n ;
n = Integer . parseInt ( s ) ;
n = 2 * n ;
s = String . valueOf ( n ) ;
```

Extraction d'une sous-chaîne

L'expression

```
s . substring ( iDeb , iFin )
```

retourne une nouvelle chaîne constituée des caractères de **s** dont les rangs sont compris entre **iDeb** et **iFin - 1**

L'expression

```
s . substring ( iDeb )
```

retourne une nouvelle chaîne constituée des caractères de **s** dont les rangs sont compris entre **iDeb** et **s . length ( ) - 1**

Exercice

Ecrire la fonction **heureEnSecondes**

qui admet un paramètre de type chaîne de caractères

de la forme `"HH:MM:SS"`

et retourne le nombre de secondes correspondant.

Par exemple

```
heureEnSecondes ( "01:02:20" )
```

retourne `3740 ( 3600 + 120 + 20 )`

Exercice

Ecrire la fonction **secondesEnHeure**

qui admet un paramètre de type entier le nombre de secondes

et retourne une chaîne de caractères de la forme

de la forme `"HH:MM:SS"`

Par exemple

```
secondesEnHeure ( 3740 )
```

retourne `"01:02:20"`

Nom d'une variable

- Le nom d'une variable est appelé identificateur.
- Un identificateur est une suite de caractères alphanumériques commençant par une lettre.
- Un identificateur ne comporte pas de caractère espace.
- On distingue les lettres majuscules et minuscules.
- Par convention, en java le nom d'une variable commence par une lettre minuscule.
- Par souci de lisibilité, il est important de donner aux variables des noms significatifs.
- Le nom d'une variable ne peut pas être égal à un mot réservé du langage, par exemple `for`, `if`, `int` ...

Les opérations sur les booléens

Les opérateurs possibles sont :

- ! non
- && et court-circuit
- & et non court-circuit
- || ou court-circuit
- | ou non court-circuit

L'opérateur non

p	! p
false	true
true	false

L'opérateur et non court-circuit

p	q	p & q
false	false	false
false	true	false
true	false	false
true	true	true

L'opérateur et court-circuit

p	q	p && q
false	?	false
true	false	false
	true	true

Si le premier opérande **p** est faux le deuxième opérande **q** n'est pas évalué (il peut être indéfini)

L'opérateur ou non court-circuit

p	q	p | q
false	false	false
false	true	true
true	false	true
true	true	true

L'opérateur ou court-circuit

p	q	p || q
true	?	true
false	false	false
	true	true

Si le premier opérande **p** est vrai le deuxième opérande **q** n'est pas évalué (il peut être indéfini)

Expressions booléennes
Opérateurs relationnels

==	égal
!=	différent
>	supérieur
<	inférieur
>=	supérieur ou égal
<=	inférieur ou égal

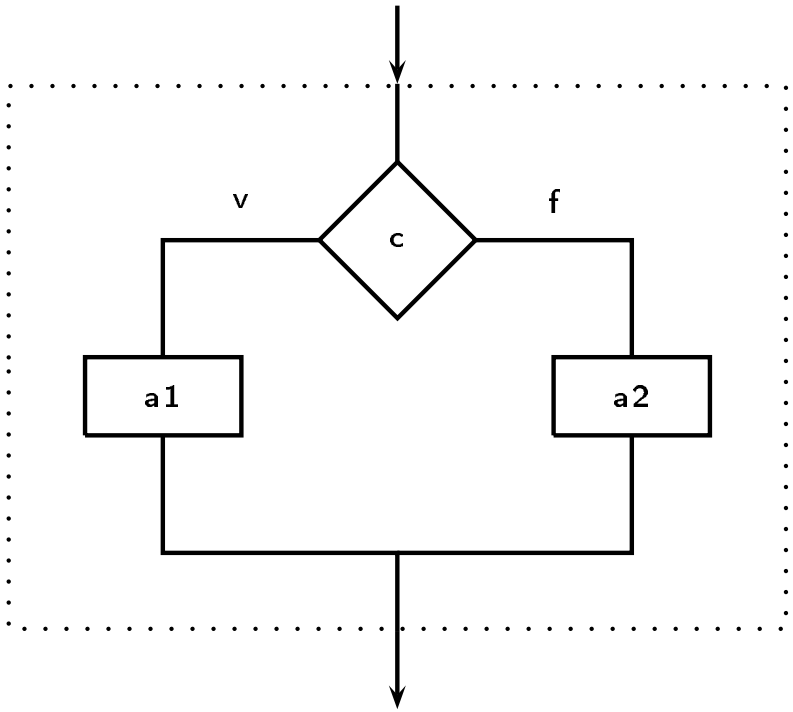
L'expression $t \in [\, t1, t2\,]$ s'écrit :

```
t >= t1 && t <= t2
```

L'expression $t \notin [\, t1, t2\,]$ s'écrit :

```
t <= t1 || t >= t2
```

Alternative




```
if ( c ) {
    a1
} else {
    a2
}
```

La condition **c** doit être une expression booléenne (dont la valeur est égale à vrai ou faux)
Si la condition est vraie, l'action **a1** est exécutée, dans le cas contraire l'action **a2** est exécutée

Déclaration ou définition d'une fonction

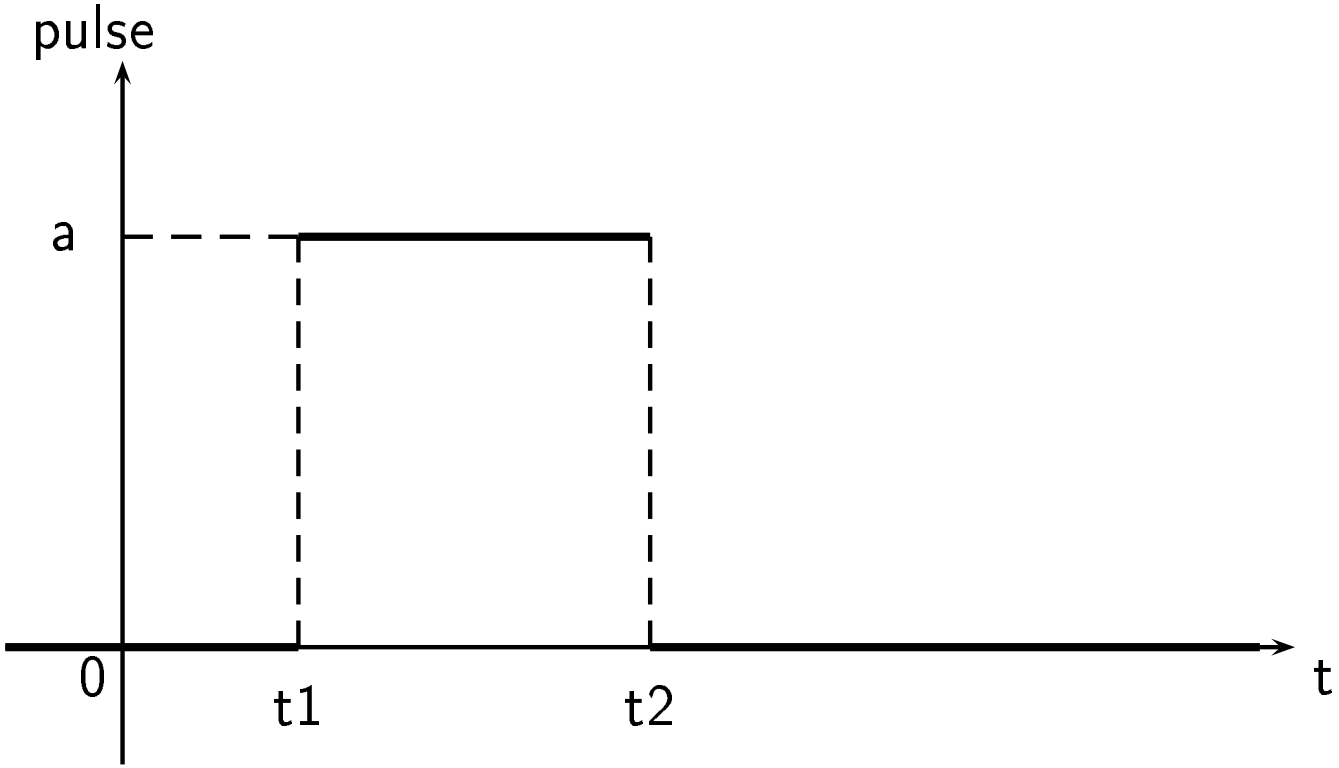
La déclaration d'une fonction (avant la fonction main) est de la forme :

```
public static typeValRet nomFct ( listParamForm ) {

    // corps de la fonction

}
```

- **typeValRet** est le type de la valeur retournée.
  - **void** indique que la fonction ne retourne rien
- le nom de la fonction **nomFct** doit indiquer clairement le traitement effectuée par la fonction.
- **listParamForm** est la liste des paramètres formels

La fonction pulse

La fonction **pulse** est définie par la représentation graphique suivante :



La fonction pulse

```
public static double pulse( double t, double t1, double t2, double a ) {


}
```

La fonction pulse

```
public static double pulse( double t, double t1, double t2, double a ) {
    if ( t >= t1 && t <= t2 ) {
        return a ;
    } else {
        return 0 ;
    }
}
```

La fonction pulse

```
public static double pulse( double t, double t1, double t2, double a ) {
    double p ;
    if ( t >= t1 && t <= t2 ) {
        p = a ;
    } else {
        p = 0 ;
    }
    return p ;
}
```

La fonction fact

La factorielle de n notée $n!$ est égale à :

$$1 * 2 * 3 * \cdots * n$$

Par convention $0! = 1$

La fonction fact

```
public static int fact ( int n ) {
    // n est un entier >= 0

}
```

La fonction fact

```
public static int fact ( int n ) {
    // n est un entier >= 0
    int f ;
    f = 1 ;
    for ( int i = 2 ; i <= n ; i = i + 1 ) {
        f = f * i ;
    }
    return f ;
}
```

La fonction fact

Attention le code suivant

```
public static int fact ( int n ) {
    // n est un entier >= 0
    int f ;
    f = 1 ;
    for ( int i = 0 ; i <= n ; i = i + 1 ) {
        f = f * i ;
    }
    return f ;
}
```

est faux, pourquoi ?

La fonction fact

Donner l'instruction qui permet d'obtenir la valeur de

$$n! + p!$$

On supposera que les variables n et p ont été déclarées et ont reçues une valeur.

La fonction fact

Donner l'instruction qui permet d'obtenir la valeur de

$$n! + p!$$

On supposera que les variables n et p ont été déclarées et ont reçues une valeur.

```
System.out.println ( fact ( n ) + fact ( p ) );
```

La fonction fact

Donner l'instruction qui permet d'obtenir la valeur de

$$n! + p!$$

On supposera que les variables n et p ont été déclarées et ont reçues une valeur.

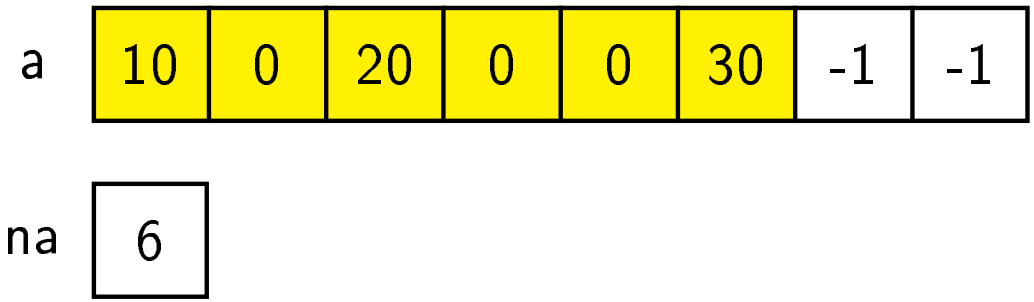
La réponse suivante

```
fact ( n ) + fact ( p );
```

est une erreur, pourquoi ?

Tableaux non remplis

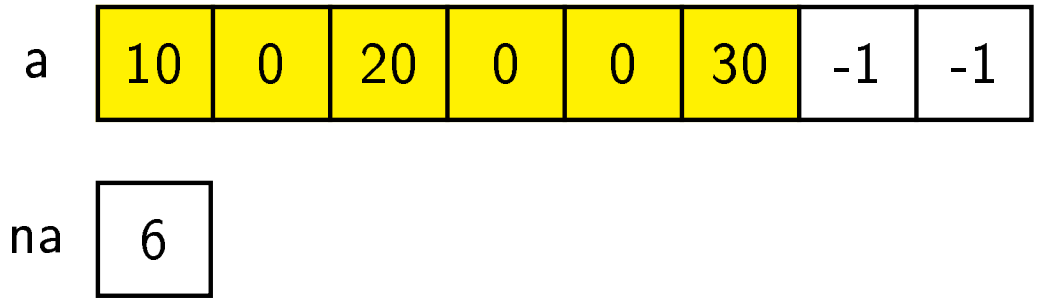
Soit le tableau non rempli **a**



na donne le nombre d'éléments du tableau **a**.

Donner les instructions qui permettent de déclarer et de donner une valeur à ces deux variables.

Tableaux non remplis




```
int [ ] a ;

a = new int [ ] { 10, 0, 20, 0, 0, 30, -1, -1 };

int na ;

na = 6 ;
```

La fonction nbZeros

Donner la fonction **nbZeros** qui retourne le nombre de zéros du tableau **a** de **na** éléments.

La fonction nbZeros

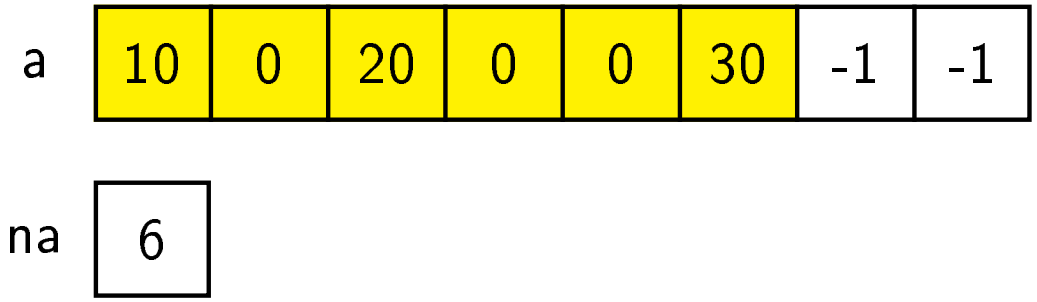
```
public static int nbZeros(int [ ] a, int na) {


}
```

La fonction nbZeros

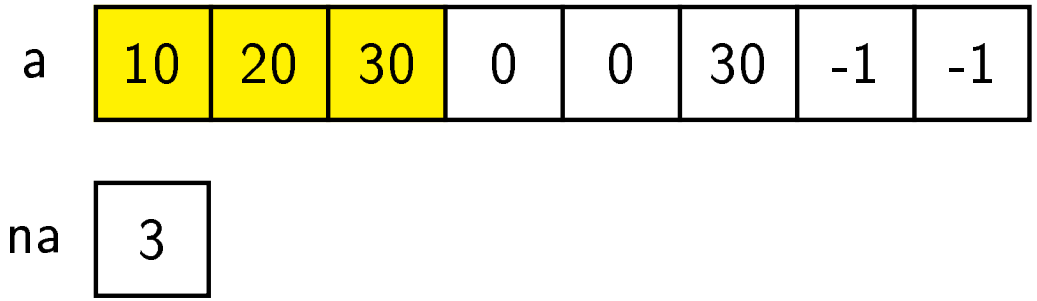
```
public static int nbZeros(int [ ] a, int na) {
    int nz ;
    nz = 0 ;
    for ( int i = 0 ; i <= na - 1 ; i=i+1 ) {
        if ( a [ i ] == 0 ) {
            nz = nz + 1 ;
        }
    }
    return nz ;
}
```

Suppression des zéros

Soit le tableau non rempli **a**



après suppression des zéros le tableau devient :



La fonction supprimeZeros

Cette fonction retourne le nombre d'éléments du tableau après suppression des zéros

```
public static int supprimeZeros(int [ ] a, int na){


}
```

Pour réaliser ce traitement on utilise deux indices

- **i** pour parcourir les éléments d'indices **0** à **na** - **1**
- **j** pour ranger les éléments non nuls

La fonction supprimeZeros

Cette fonction retourne le nombre d'éléments du tableau après suppression des zéros

```
public static int supprimeZeros(int [ ] a, int na){
    int j ;
    j = 0 ;
    for ( int i = 0 ; i <= na - 1 ; i = i - 1 ) {
        if ( a [ i ] != 0 ) {
            a [ j ] = a [ i ] ;
            j = j + 1 ;
        }
    }
    return j ;
}
```

Pour réaliser ce traitement on utilise deux indices

- **i** pour parcourir les éléments d'indices **0** à **na - 1**
- **j** pour ranger les éléments non nuls

La fonction supprimeZeros

L'instruction qui permet de supprimer les zéros du tableau **a** de **na** éléments est :

```
na = supprimeZeros ( a , na ) ;
```

Tri d'un tableau de chaînes de caractères

Le tableau de chaînes de caractères :

	0	1	2	3	4
mots	Python	Java	Ada	C	Perl

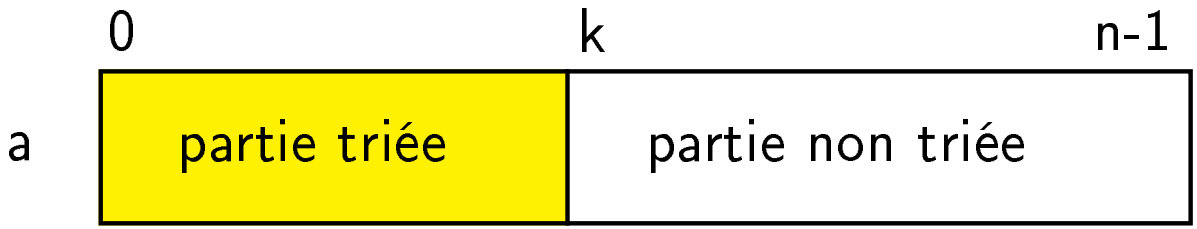
après le tri le tableau précédent devient :

	0	1	2	3	4
mots	Ada	C	Java	Perl	Python

Tri par sélection

Soit **a** un tableau et **n** son nombre d'éléments.

n est égal à **a.length** $-$ **1**



A l'étape **k** on recherche (séléctionne) le plus petit élément de la partie non triéé et on l'échange avec le premier élément (d'indice k) de cette partie non triée.

Tri par sélection (suite)

L'algorithme du tri par séléction est donc :

pour i **de** 0 **à** n - 2 **faire**

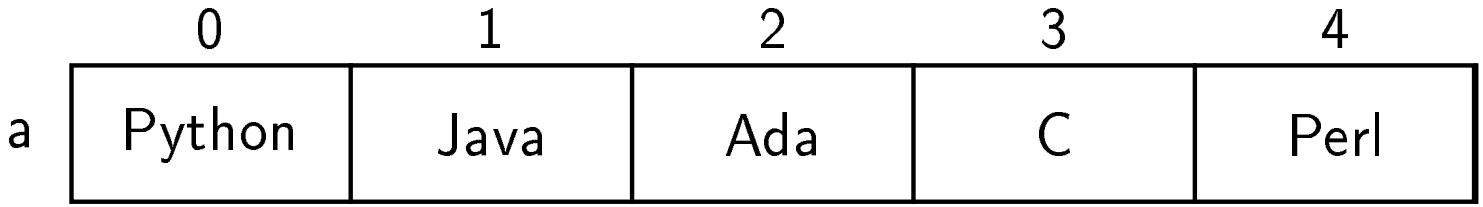
// déterminer l'indice **iMin** du plus petit élément
// de **a [k : a . length - 1]**

// échanger les éléments d'indice **k** et **iMin**

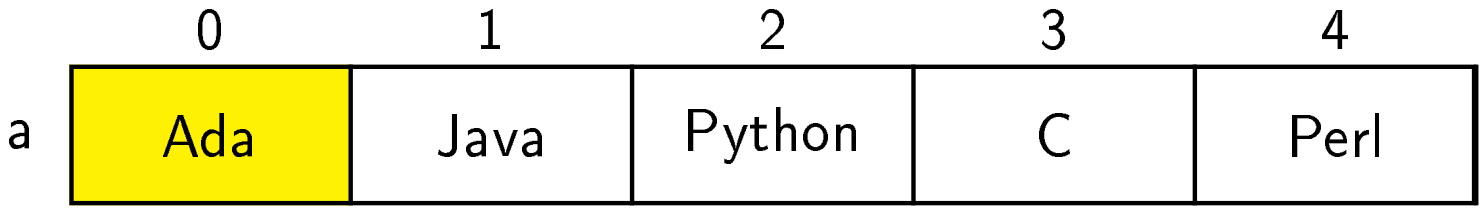
fpour

Tri par sélection (suite)

Pour le tableau a



à l'étape **k=0** on trouve **iMin=2**, le tableau a devient :



Tri par sélection (suite)

	0	1	2	3	4
a	Ada	Java	Python	C	Perl

à l'étape **k=1** on trouve **iMin=3**, le tableau a devient :

	0	1	2	3	4
a	Ada	C	Python	Java	Perl

à l'étape **k=2** on trouve **iMin=3**, le tableau a devient :

	0	1	2	3	4
a	Ada	C	Java	Python	Perl

Tri par sélection (suite)

	0	1	2	3	4
a	Ada	C	Java	Python	Perl

à l'étape **k=3** on trouve **iMin=4**, le tableau a devient :

	0	1	2	3	4
a	Ada	C	Java	Perl	Python

Il ne reste plus qu'un élément supérieur à tous ceux qui le précèdent, le tableau **a** est donc complètement trié.

	0	1	2	3	4
a	Ada	C	Java	Perl	Python

Execice 1

Ecrire la fonction **tri** qui permet de trier par ordre croissant le tableau **a** de chaînes de caractères passé en paramètre.

Cette fonction doit utiliser les deux fonctions **indMin** et **echange**

Excercice 1 (suite)

```
package tp12 ;

public class Tri {
    public static void echange ( String [ ] a, int i1 , int i2 ) {

    }

    public static int indMin ( String [ ] a , int iDeb ) {

    }

    public static void tri ( String [ ] a ) {

    }

    public static void main ( String [ ] args ) {
        String [ ] mots ;
        mots = new String [ ]
                    { "Python" ,"Java" ,"Ada" , "C" , "Perl" } ;

        Tableaux.afficher( mots );
        tri ( mots ) ;
        Tableaux.afficher( mots );
    }
}
```

Egalité de deux chaînes de caractères

Il n'est pas possible d'utiliser les opérateurs d'égalité ou d'inégalité avec les chaînes de caractères
L'expression

```
s1 . equals ( s2 )
```

retourne vrai si les deux chaînes sont égales et faux dans le cas contraire.

Egalité de deux chaînes de caractères (suite)

```
    ...
s1 = "ABC" ;
s2 = "abc" ;
System.out.println( s1.equals ( s2 ) ) ;
  // -> false
System.out.println( s1.equalsIgnoreCase ( s2 ) ) ;
  // -> true
```

Ordre lexicographique

La chaîne **s1** précède la chaîne **s2** (**s1** $\prec$ **s2**) **ssi**

- **s1** est un préfixe de **s2**

  bal $\prec$ ballon

- Le premier caractère de **s1** qui ne coïncide pas avec le caractère de même range de **s2** est inférieur à ce dernier

  ballon $\prec$ bas

Comparaison de deux chaînes de caractères

L'expression

```
s1 . compareTo ( s2 )
```

$$\text{retourne une valeur} \begin{cases} < 0 & \text{si} \quad s1 \prec s2 \\ 0 & \text{si} \quad s1 \text{ est égal à } s2 \\ > 0 & \text{si} \quad s1 \succ s2 \end{cases}$$

L'expression

```
s1 . compareToIgnoreCase ( s2 )
```

effectue la même comparaison sans tenir compte des majuscules et minuscules.

Recherche séquentielle d'une valeur dans un tableau non rempli

Ecrire la fonction **appartient** qui admet trois paramètres

- **a** un tableau de chaînes de caractèes dans lequel s'effectue la recherche
- **nbElts** entier égal au nombre d'éléments du tableau précédent
- **valRech** chaîne de caractères valeur recherchée dans le tableau

et qui retourne **vrai** si la valeur recherchée **valRech** appartient au tableau **a** et **faux** dans le cas contaire.

La fonction appartient

```
public static boolean appartient( String [ ] a, int nbElts, String valRech) {



}
```

Elimination des doublons

```
public static int elimDoublons ( String [ ] a, int nbElts ) {



}
```

Compléments sur les littéraux entiers

Par défaut les littéraux entiers sont écrits en décimal.

Ils peuvent aussi être écrits en hexadécimal ou en octal s'ils commencent par `0x` ou `0`

25	
0x19	25 en hexadécimal (16 + 9)
031	25 en octal (3 * 8 + 1)

Rappels sur l'algèbre de Boole (1)

L'opérateur non

x	$\overline{x}$
0	1
1	0

Rappels sur l'algèbre de Boole (2)

L'opérateur et

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

$$x \cdot 0 = 0$$

$$x \cdot 1 = x$$

Rappels sur l'algèbre de Boole (3)

L'opérateur ou

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

$$x + 0 = x$$

$$x + 1 = 1$$

Rappels sur l'algèbre de Boole (4)

L'opérateur ou exclusif

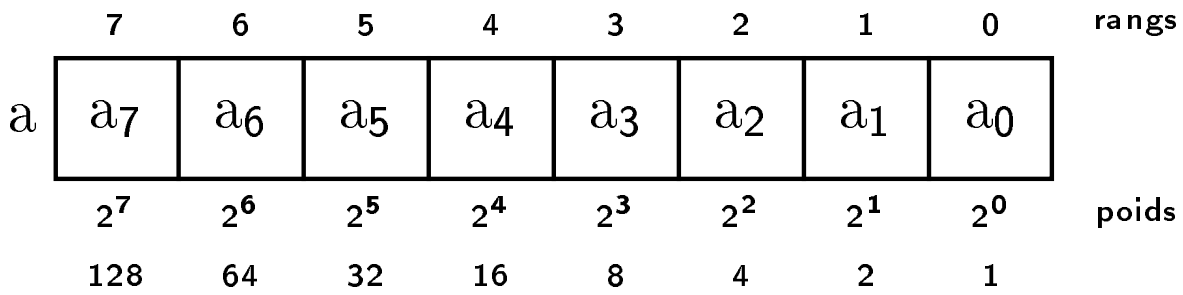
x	y	x $\oplus$ y
0	0	0
0	1	1
1	0	1
1	1	0

$$x \oplus 0 = x$$

$$x \oplus 1 = \overline{x}$$

Le système binaire

Soit **a** un octet :



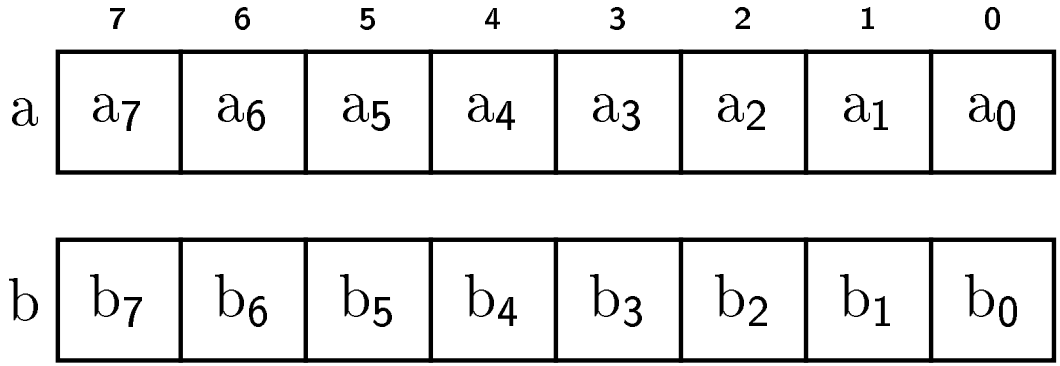
Le bit de rang i a pour poids 2^i

Les opérateurs bits à bits

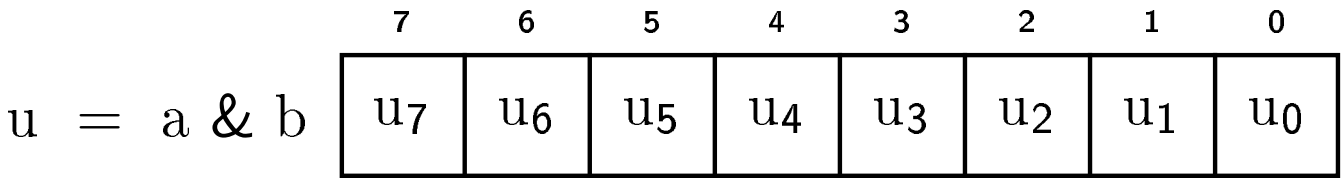
&	et bits à bits
|	ou bits à bits
^	ou exclusif bits à bits
~	complément

Le et bits à bits &

Soient **a** et **b** deux octets (ou plus généralement des mots de n bits)



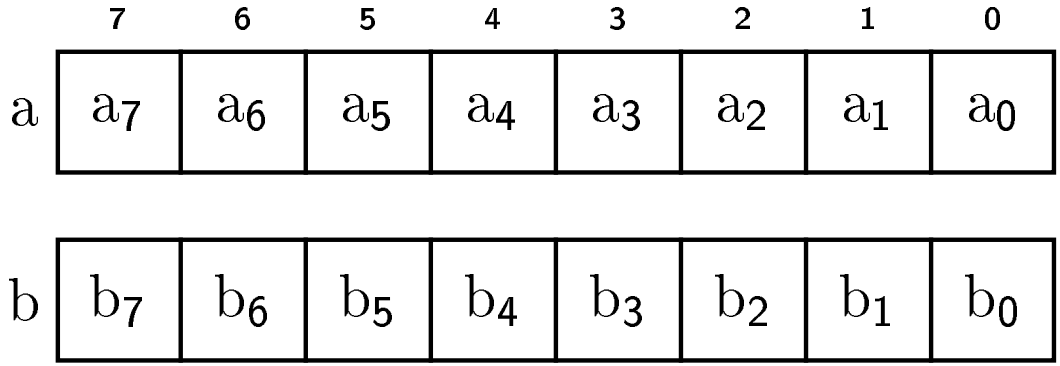
Le **et** bits à bits entre les octets **a** et **b** est :



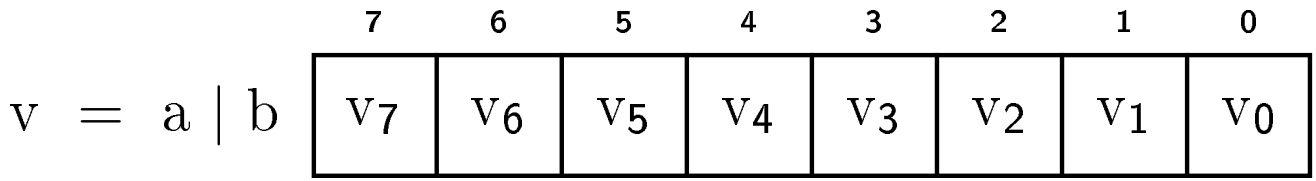
avec $u_i = a_i \cdot b_i$

Le ou bits à bits |

Soient **a** et **b** deux octets (ou plus généralement des mots de n bits)



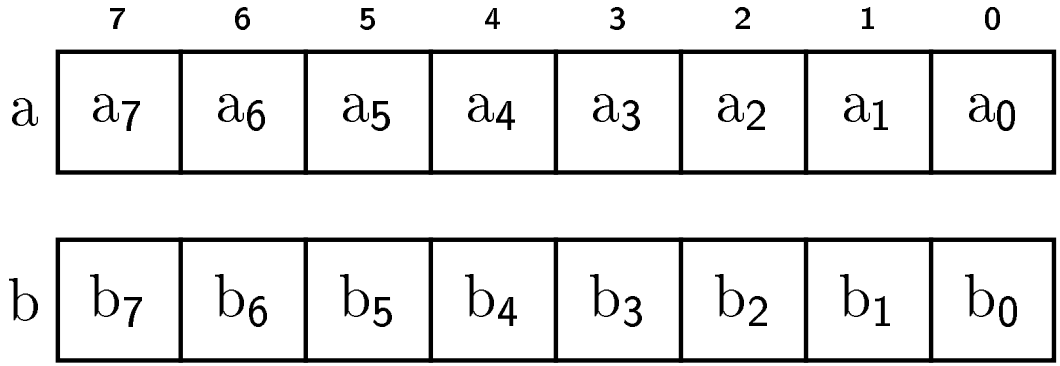
Le **ou** bits à bits entre les octets **a** et **b** est :



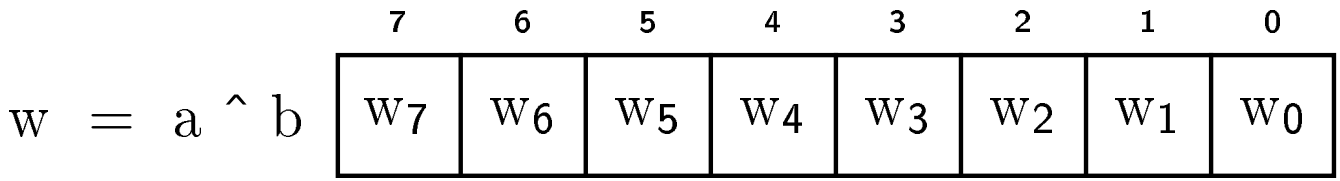
avec $v_i = a_i + b_i$

Le ou exclusif bits à bits ^

Soient **a** et **b** deux octets (ou plus généralement des mots de n bits)



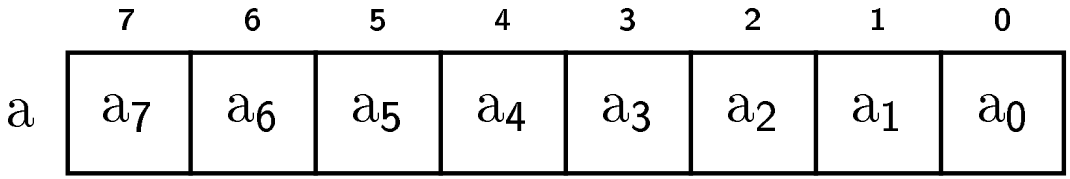
Le **ou** exclusif bits à bits entre les octets **a** et **b** est :



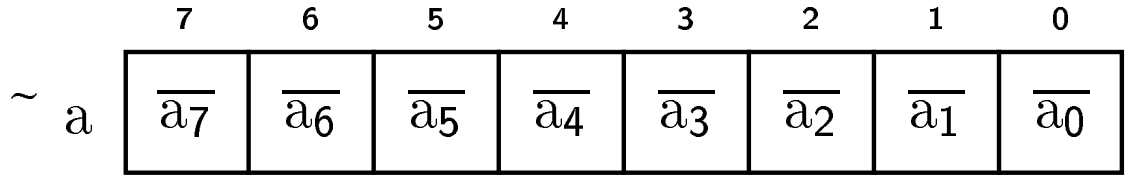
avec $w_i = a_i \oplus b_i$

Le complément ~

Le complément de l'octet **a**

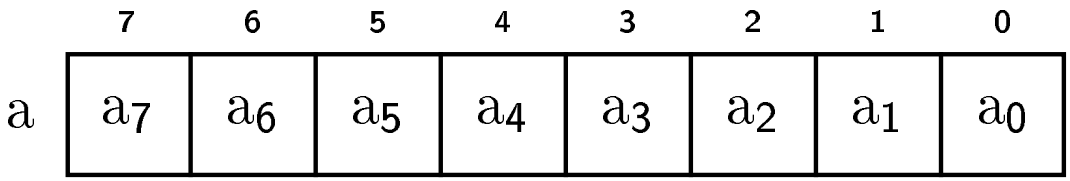


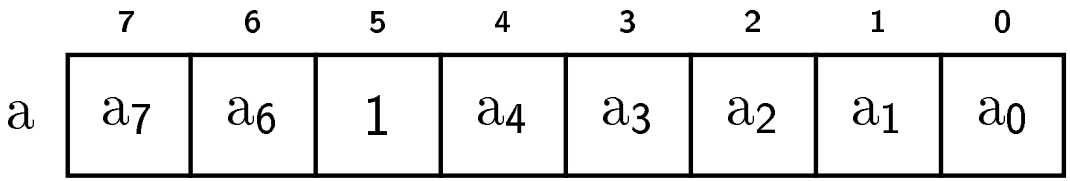
est l'octet



Exercice 1

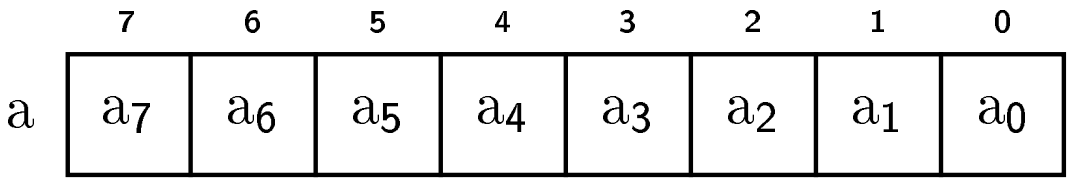
Donner l'instruction qui permet de forcer à 1 le bit de rang 5 de l'octet **a**

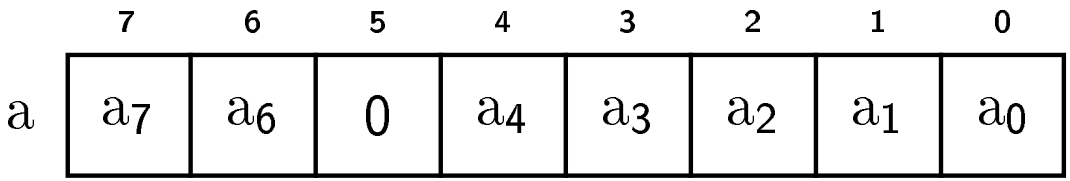




Exercice 2

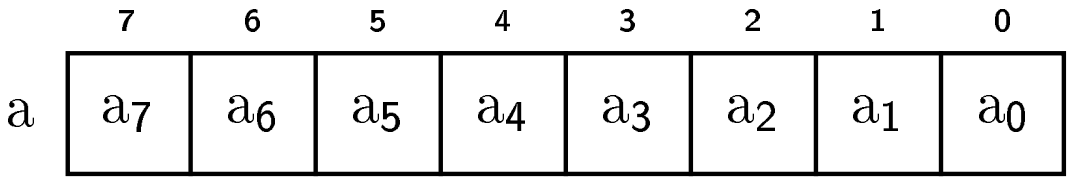
Donner l'instruction qui permet de forcer à 0 le bit de rang 5 de l'octet **a**

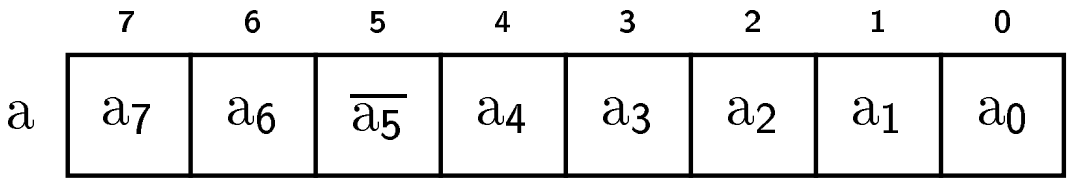




Exercice 3

Donner l'instruction qui permet de complémenter le bit de rang 5 de l'octet **a**



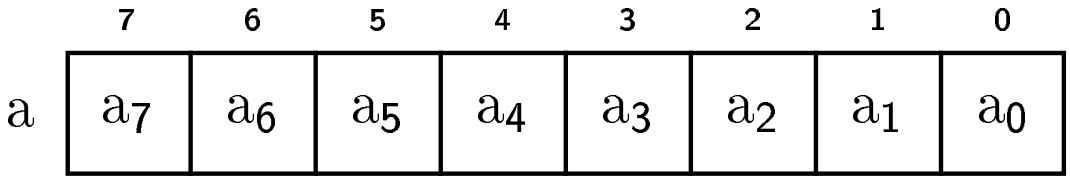


Les opérateurs de décalage

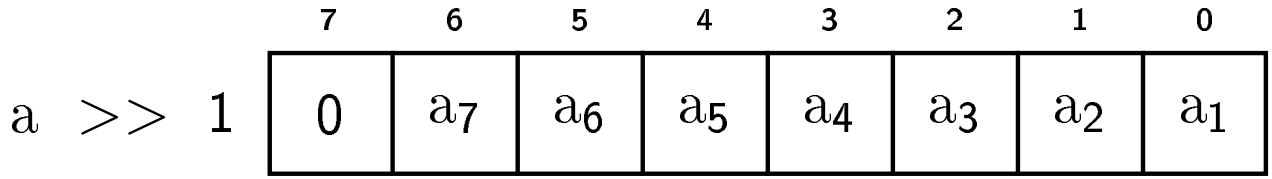
>>	décalage à droite
<<	décalage à gauche

Décalage à droite (1)

Le décalage à droite d'une position de l'octet **a**

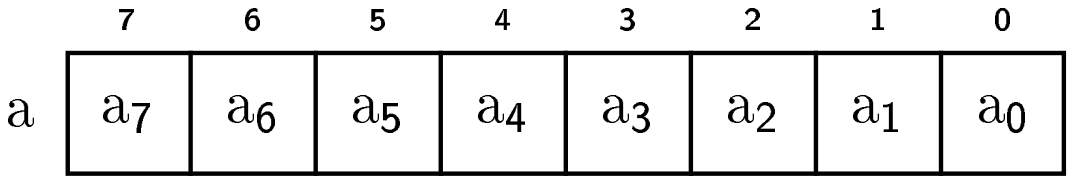


est :

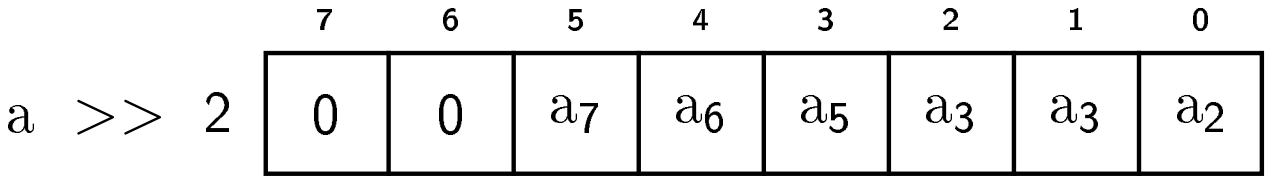


Décalage à droite (2)

Le décalage à droite de deux positions de l'octet **a**

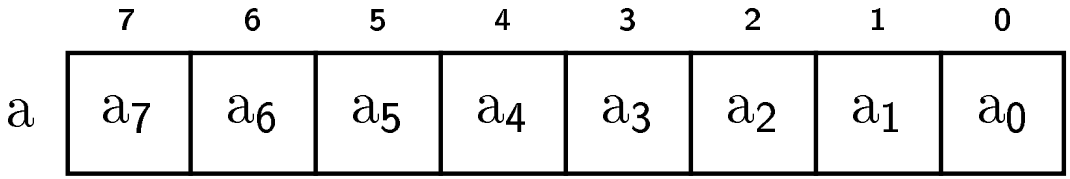


est :

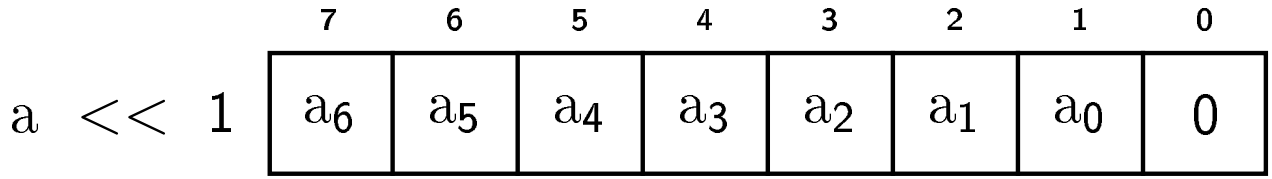


Décalage à gauche (1)

Le décalage à gauche d'une position de l'octet **a**

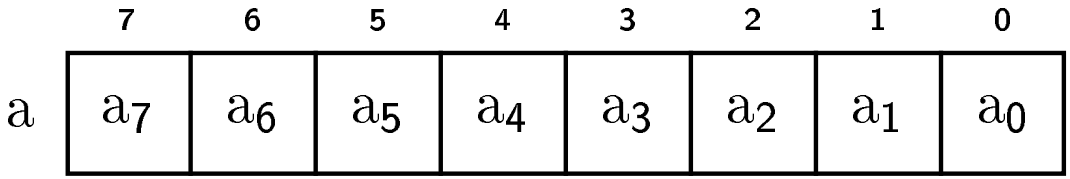


est :

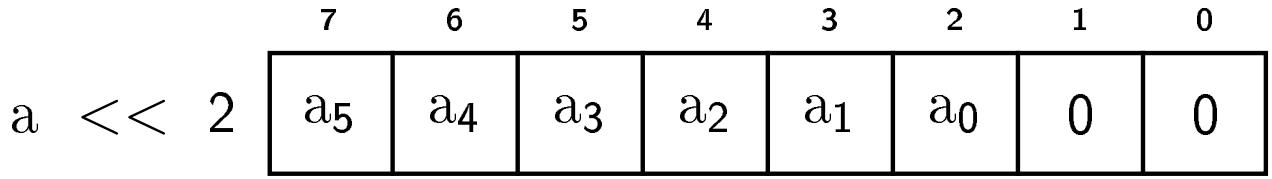


Décalage à gauche (2)

Le décalage à gauche de deux positions de l'octet **a**



est :



La fonction quartetPoidsFaible

Ecrire la fonction **quartetPoidsFaible** qui retourne le quartet de poids faible de l'octet passé en paramètre.

```
quartetPoidsFaible ( 25 ) retourne   9
                                ( 25 = 0x19 )
```

```
public static int quartetPoidsFaible( int n ){

}
```

La fonction quartetPoidsFort

Ecrire la fonction **quartetPoidsFort** qui retourne le quartet de poids fort de l'octet passé en paramètre.

```
quartetPoidsFort ( 25 ) retourne  1
                          ( 25 = 0x19 )
```

```
public static int quartetPoidsFort( int n ){

}
```

La fonction quartetToHexa (1)

La fonction **quartetToHexa** retourne un caractère égal au chiffre hexadécimal du quartet passé en paramètre.

quartet	car hexa
0	'0'
...	...
9	'9'
10	'a'
11	'b'
12	'c'
13	'd'
14	'e'
15	'f'

La fonction quartetToHexa (2)

```
quartetToHexa (  5 ) retourne  le caractère '5'

quartetToHexa ( 10 ) retourne  le caractère 'a'
```

```
public static char quartetToHexa(int quartet ){

}
```

La fonction octetToHexa

La fonction **octetToHexa** retourne une chaîne de caractères égale au nombre hexadécimal de l'octet passé en paramètre.

```
octetToHexa ( 25 )
    retourne  la chaîne de caractères "19"

octetToHexa ( 11 )
    retourne  la chaîne de caractères "0b"
```

```
public static String octetToHexa(int octet ){


}
```

La fonction setBit

La fonction **setBit**

```
public static int setBit ( int b , int i ) {

}
```

retourne la valeur l'octet **b** avec le bit de rang **i** forcé à 1.

On suppose que

- la valeur de l'octet **b** est comprise entre 0 et 255
- la valeur du rang **i** est comprise entre 0 et 7

La fonction resetBit

La fonction **resetBit**

```
public static int resetBit ( int b , int i ) {

}
```

retourne la valeur l'octet **b** avec le bit de rang **i** forcé à 0.

On suppose que

- la valeur de l'octet **b** est comprise entre 0 et 255
- la valeur du rang **i** est comprise entre 0 et 7

La fonction compBit

La fonction **compBit**

```
public static int compBit ( int b , int i ) {

}
```

retourne la valeur l'octet **b** avec le bit de rang **i** complémenté.

On suppose que

- la valeur de l'octet **b** est comprise entre 0 et 255
- la valeur du rang **i** est comprise entre 0 et 7

Utilisation de la classe complexe

Déclaration d'un nombre complexe

```
Complexe z ;
```

Constuction d'un nombre complexe dont

- la partie réelle est **a**
- et la partie imaginaire **b**

```
z = new Complexe ( a , b ) ;
```

où a et b sont deux réels.

Module et argument

```
z . mod ( )
```

retourne le module du nombre complexe **z**

```
z . arg ( )
```

retourne l'argument en radians du nombre complexe **z**

```
z . argD ( )
```

retourne l'argument en degrés du nombre complexe **z**

Exercice 1

Ecrire un programme qui crée le nombre complexe $1 + \mathrm{i}$ et affiche son module et son argument en degrés.

Saisie et affichage

L'instruction

```
z . saisir ( )
```

permet de saisir la partie réelle et imaginaire du nombre complexe **z**.

Attention ce dernier doit être construit.

L'instruction

```
z . afficher ( )
```

permet d'afficher la partie réelle et la partie imaginaire du nombre complexe **z**.

Exercice 2

Ecrire un programme qui

- crée le nombre complexe
- saisit sa partie réelle et imaginaire
- affiche sa partie réelle et sa partie imaginaire puis son module et son argument en degrés.

Partie réelle et imaginaire

```
z . re ( )
```

retourne la partie réelle du nombre complexe z

```
z . im ( )
```

retourne partie imaginaire du nombre complexe z

Exercice 3

Ecrire un programme qui

- crée deux nombres complexes **z1** et **z2**
- saisit leurs parties réelles et imaginaires
- calcule le nombre complexe $\mathbf{z3 = z1 + z2}$
- affiche sa partie réelle et sa partie imaginaire puis son module et son argument en degrés.

Exercice 4

Ecrire un programme qui

- crée deux nombres complexes **z1** et **z2**
- saisit leurs parties réelles et imaginaires
- calcule le nombre complexe $\mathbf{z4 = z1 * z2}$
- affiche sa partie réelle et sa partie imaginaire puis son module et son argument en degrés.

Addition et multiplication

```
z1 . add ( z2 )
```

retourne le nombre complexe **z1 + z2**

L'instruction

```
z1 . mul ( z2 )
```

retourne le nombre complexe **z1 * z2**

Exercice 5

Ecrire un programme qui

- crée deux nombres complexes **z1** et **z2**
- saisit leurs parties réelles et imaginaires
- calcule le nombre complexe $\mathbf{z3 = z1 + z2}$
- calcule le nombre complexe $\mathbf{z4 = z1 * z2}$
- affiche les parties réelles et imaginaires de **z3** et **z4**

Exercice 6

Ecrire un programme qui

- crée deux nombres complexes **z1** et **z2**
- saisit leurs parties réelles et imaginaires
- calcule le nombre complexe

$$\mathbf{z} = \frac{\mathbf{z1} * \mathbf{z2}}{\mathbf{z1} + \mathbf{z2}}$$

- affiche la partie réelle et imaginaire de **z**