ITI1520 – Introduction à l'informatique I
Cahier d'exercices
Automne 2010

Table de matière

1. Introduction

Ce cahier d'exercices contient toutes les exercices qui seront étudiées et travaillées en classe. Amenez-le à tous vos cours magistraux. Les sections 1 à 4 résume les concepts informatiques qui sont utiles pour compléter les exercices ainsi que répondre aux questions des devoirs. Elles comprennent :

- **Modèle de programmation d'ordinateur** – le modèle de programmation est une représentation des composantes principales utilisé dans l'exécution de programmes. Le modèle permet de comprendre comment un programme est exécuté. Il est utilisé avec la majorité des exercices du cahier. Une description du modèle est fournie dans la section 2. Cette section donne aussi deux exemples de modèle vides pour vos études et devoirs.
- **Algorithmes prédéfinis –** Un nombre d'algorithmes prédéfinis « standards » sont disponible pour le développement de vos propres algorithmes. Ils représentent des fonctions standards disponibles dans un système informatique tel que la lecture du clavier et l'affichage à la console de terminal.
- **Structure de programme, Concepts d'algorithme et de programmation java** – Cette section donne une table qui résume plusieurs concepts et structures de programmation étudiés en class et durant les sessions de laboratoire. La serve sert de référence rapide.

2. Modèle de programmation

Les deux composantes principales servant à l'exécution d'un programme informatique sont la mémoire et l'unité centrale de traitement (UCT). Premièrement, la mémoire contient les instructions machines du programme – ces instructions machines sont normalement créées à partir des langages haut-niveau tel que le C et le Java (avec des compilateurs)

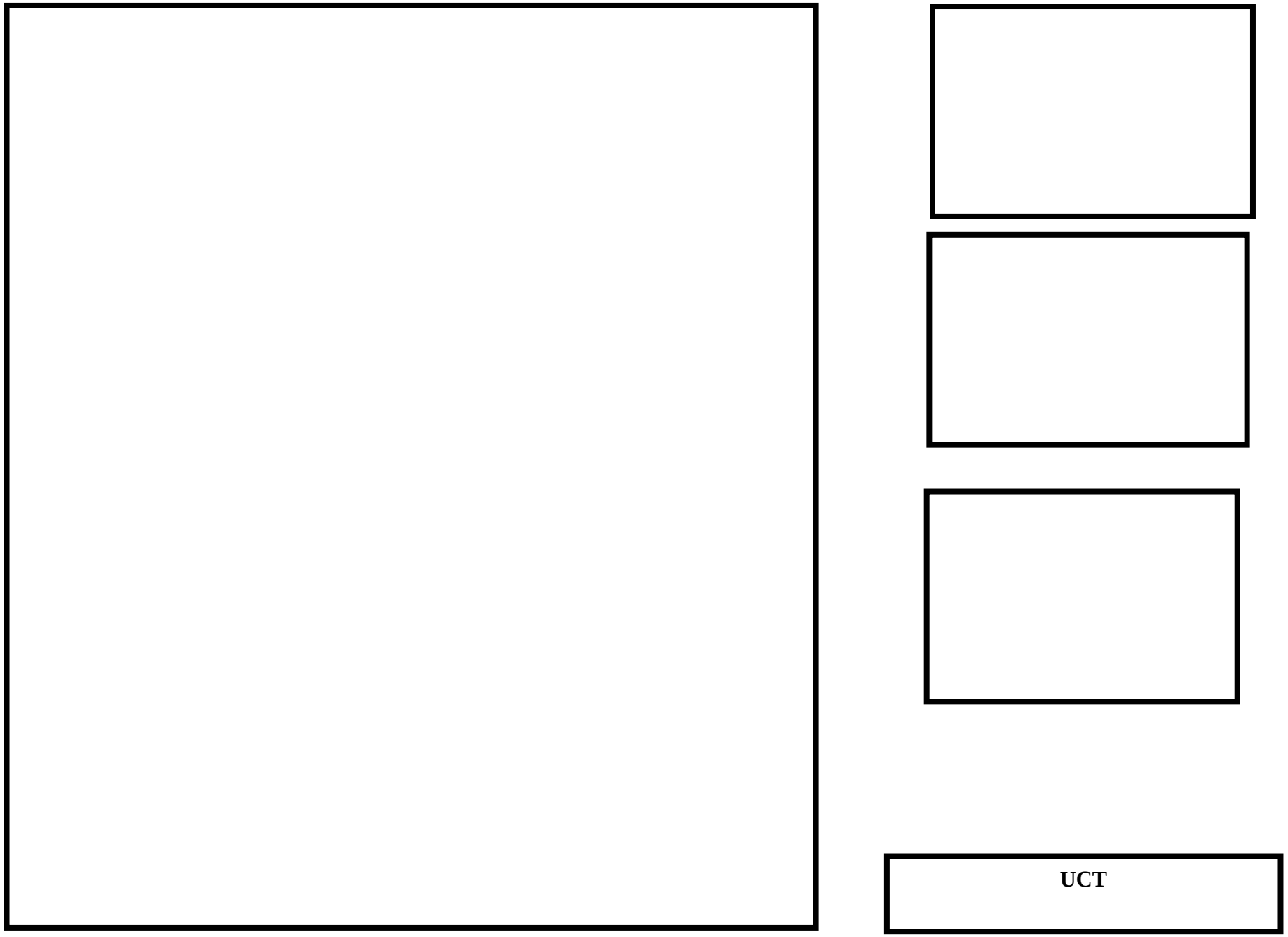
Les instructions machines manipulent des variables, qui se retrouvent aussi dans la mémoire. Essentiellement l'UCT lit et écrit des valeurs dans les variables (i.e. les emplacements en mémoire où se retrouvent les variables). Les opérations tel qu'addition, incrémentation et des testes, sont normalement complétées à l'intérieur de l'UCT.

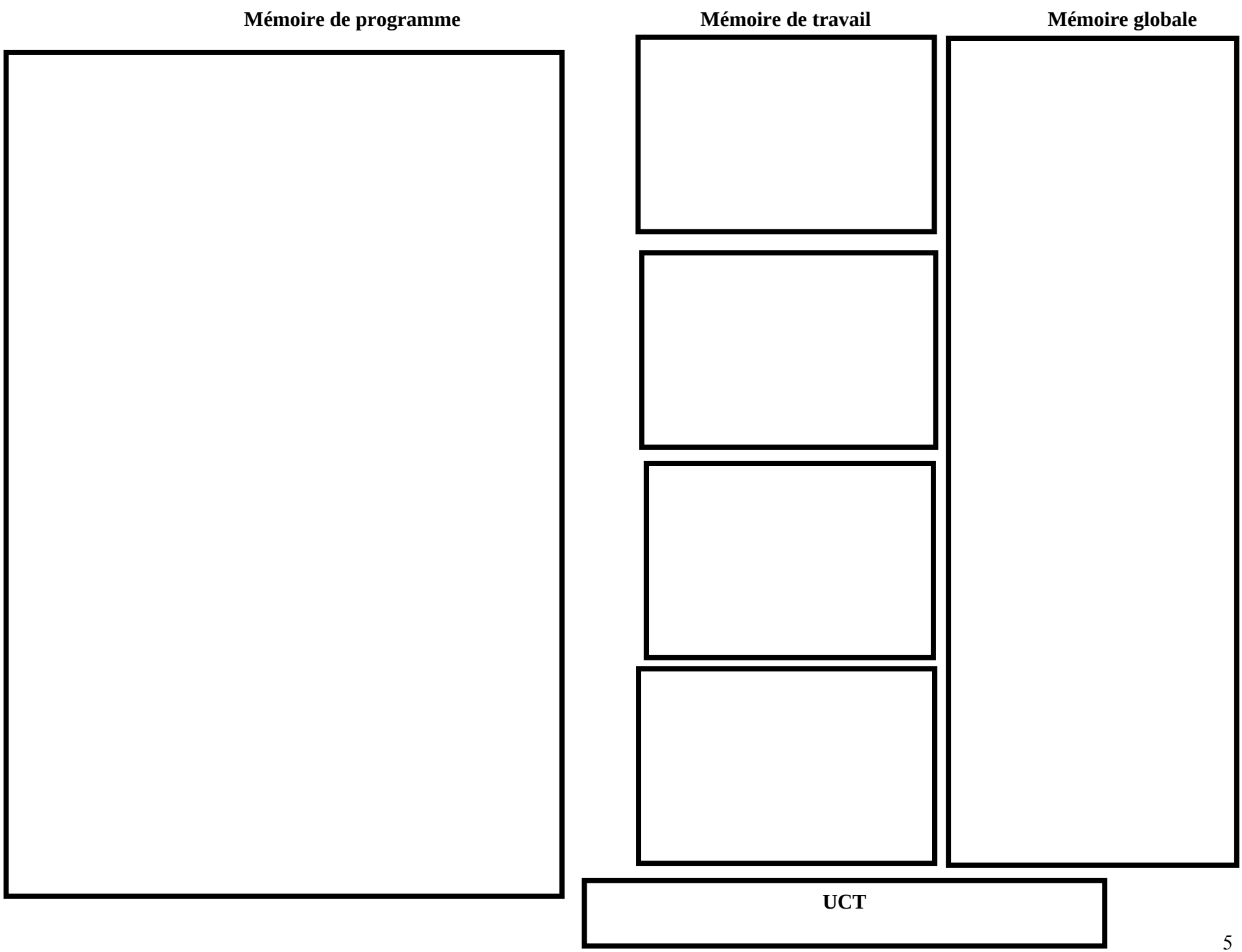
La mémoire de l'ordinateur contient une séquence d'octets (valeurs de 8 bits; le bit est un chiffre qui prend la valeur de 0 ou de 1, et l'octet contient 8 bits). Donc un ordinateur qui contient 3 gigaOctets contient 3 x 1024 x 1024 x 1024 octets, c'est-à-dire, au-delà de 3 milliards d'octets.

Chaque octet à une adresse. Une adresse est simplement un numéro qui réfère à un octet en mémoire. Donc quand un UCT veut exécuter une instruction, il lit l'instruction avec sont adresse (l'UCT maintient l'adresse des instructions pour les exécuter en séquence). Les instructions contiennent les adresses des variables à manipuler.

La mémoire utilisé par un programme est normalement divisé en régions – une partie contient les instructions du programme a exécuter, une partie est utilisé pour contenir les variables durant l'exécution de sous-programmes (cette mémoire est récupérée à la fin de l'exécution du sous-programme pour être réutilisée par l'exécution d'autres sous-programmes), et une autre partie est utilisé pour stocker des variables et autre structures de données qui sont disponible durant l'exécution complète du programme (par exemple pour stocker des tableaux informatiques et des objets). Le modèle de programmation comprend chacune de ces parties comme la **mémoire de programme**, la **mémoire de travail,** et la **mémoire globale** respectivement. Notez que la mémoire de travail est divisée en plusieurs morceaux pour monter comment cette mémoire est « réserver » pour l'exécution de différents sous-programmes.

Les deux pages suivantes donnent des modèles de programmation vides pour votre utilisation durant vos études et vos devoirs. La première page ne comprend pas la mémoire globale est pourra être utilisé durant presque toute la première moitié du cours puisque la mémoire globale sera utilisé après que les tableaux informatiques seront introduites.


Mémoire de programme
Mémoire de travail
UCT


Mémoire de programme
Mémoire de travail
Mémoire globale
UCT

3. Algorithmes prédéfinis

Les algorithmes suivants peuvent être utilisés pour le développement d'algorithmes. La majorité représente l'entrée et la sortie pour interagir avec l'utilisateur. Les trois derniers algorithmes représentent d'autre tâches disponibles dans l'ordinateur (ou langage).

AfficheLigne(<liste de paramètres>)
: Imprime une chaîne à la console (le curseur est ensuite placé à la prochaine ligne). La liste de paramètres est une liste d'items séparés de virgules qui peut être une chaîne de caractères (e.g. « une chaîne »), ou un nom de variable.

Affiche (<liste de paramètres >)
: Imprime une chaîne à la console (le curseur est laissé à la fin de la chaîne imprimée). La liste de paramètres est une liste d'items séparés de virgules qui peut être une chaîne de caractères (e.g. « une chaîne »), ou un nom de variable.

VarNbr ← LireRéel() pour lire un nombre réelle du clavier

VarEnt ← LireEntier()
: Lit un nombre entier du clavier

VarBool ← LireBooléen()
: Lit "true" ou "false"; il donne un Booléen come résultat.

VarCar ← LireCaractère()
: Lit un seul caractère du clavier.

(TblEntier, N) ← LireEntiersLigne
: Lit une ligne d'entiers, extrait les entiers de la ligne et les ajoute à un tableau (notez que TblEntier est une variable de référence au tableau). N représente le nombre d'éléments dans le tableau.

(TblRéel, N) ← LireRéelsLigne
: Lit une ligne d'entiers, extrait les valeurs réelles de la ligne et les ajoute à un tableau (notez que TblRéel est une variable de référence au tableau). N représente le nombre d'éléments dans le tableau.

(TblCar, N) ← LireCarsLigne
: Lit une ligne d'entiers, et ajoute les caractères de la ligne à un tableau (notez que TblCar est une variable de référence au tableau). N représente le nombre d'éléments dans le tableau.

VarChn ← LireLigne()
: Lit une ligne d'entrée, i.e, une chaîne du clavier.

VarNbr ← Aléatoire()
: Génère une valeur aléatoire plus grande ou égale à 0.0 et plus petit que 1.0.

uneVarRefTbl ← CréerTableau(L)
: Crée un tableau ayant L positions ayant des valeurs inconnues. La variable uneVarRefTbl est une variable de référence à laquelle est affectée l'adresse du tableau.

unVarRefMat ← CréerMatrice(R, C)
: Crée une matrice ayant R rangées fois C colonnes d'éléments ayant des valeurs inconnue. La variable uneVarRefMat est une variable de référence à laquelle est affectée l'adresse d'une tableau dont les éléments sont aussi des variables de références à chaque rangée (donc unVarRefMat[0] est une variable de référence qui réfère à un tableau représentant la première rangée de la matrice).

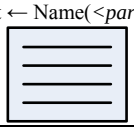
4. Sommaire de structure de programme, concepts d'algorithmes et concepts de programmation Java

Bloc d'instruction

```
{
   <instruction>
   <instruction>
        .
        .
}
```

Sous-programme

…
Result ← Name(<*paramList*>)



```
public static <type> name(<paramList>)
{
   <instruction>
   <instruction>
        .
        .
}
```

Instructions

Opération

( ) *(expression)* [ ] *(index tableau)* . (membre objet)
+ – *(valeur positive / négative)* NON
* / MOD *(multiplication, division, et modulo)*
+ - *(addition et soustraction de valeurs, concaténation)*
< > ≥ ≤ *(comparaison)*
= ≠ *(comparaison)*
ET *(opérateur logique)*
OU *(opérateur logique)*
← *(affectation à une variable)*

( ) *(expression)* [ ] *(index tableau)* . *(membre objet)*
+ – *(valeur positive/négative)* ! *(NON)*
* / % *(multiplication, division, et modulo)*
+ - *(addition et soustraction de valeurs, concaténation)*
< > >= <= *(comparaison)*
== != *(comparaison)*
&& *(opérateur logique ET)*
|| *(opérateur logique OU)*
= *(affectation à une variable)*

Expression simple

z ← x + y

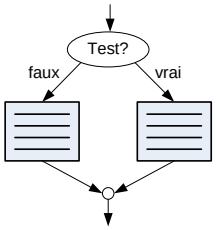
```
z = x + y;
```

Appel au sous-programme

moyenne ← analyseRésultats (score1, score2, score3)

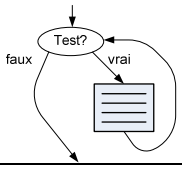
```
moyenne = analyseRésultats(score1,
             score2, score3);
```

Instruction de branchement



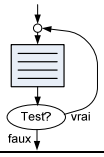

```
if (<expression logique>)
{
   <instruction>
   <instruction>
        .
        .
}
else
{
   <instruction>
   <instruction>
        .
        .
}
```

Instruction de boucle pré-test



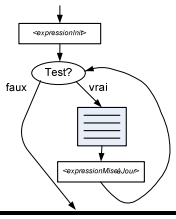

```
while(<expression logique>)
{
   <instruction>
   <instruction>
        .
        .
}
```

Instruction de boucle post-test




```
do
{
   <instruction>
   <instruction>
        .
        .
}
while(<expression logique>);
```

Instruction de boucle alternative (boucle **for** de Java)




```
for(<expressionInit>;
    <test>;
    <expressionMisàJour>)
{
   <instruction>
   <instruction>
        .
        .
}
```

Types primitifs

int	entiers
double	valeur réelle
char	caractère
boolean	valeur booléenne

Tableaux / Matrices

```
// Tableau
Type [] nonvar;
nomvar = new Type[longueur];

// Matrice
Type [][] nomvar;
nomvar = new Type[#Rang][#Cols];

// Type – tout type de variable
// (int,double, etc.) ou nom de
//  classe
// nomvar – nom de variable
// longueur – nombre d'éléments
// #Rang – nombre de rangées
// #Cols – nombre de colonnes
```

Classes / objets

```
class NomClasse
{
   // Attributs

   // Constructeur
   public NomClasse(listeParamètres)
   {
   }

   // Méthodes
   .
   .
   .
}
// NomClasse – nom valide
// mot clefs pour la
// déclaration des attributs
// et les méthodes
public – accessible par tous
private – accessible par class
          seulement
static – variable ou méthode de
         classe
// Variable de référence
NomClasse nomvar;
// Création de l'objet
nomvar = new NomClasse();
```

Méthodes/classes disponibles

Méthodes ITI1520:

```
ITI1520.readInt( )
ITI1520.readDouble( )
ITI1520.readChar( )
ITI1520.readBoolean( )
ITI1520.readDoubleLine( )
ITI1520.readIntLine( )
ITI1520.readCharLine( )
ITI1520.readString( )
```

Méthodes Scanner:

```
Scanner keyboard =
          new Scanner(System.in);
keyboard.nextInt( )
keyboard.nextDouble( )
keyboard.nextBoolean( )
keyboard.nextLine( )
```

Méthodes System.out:

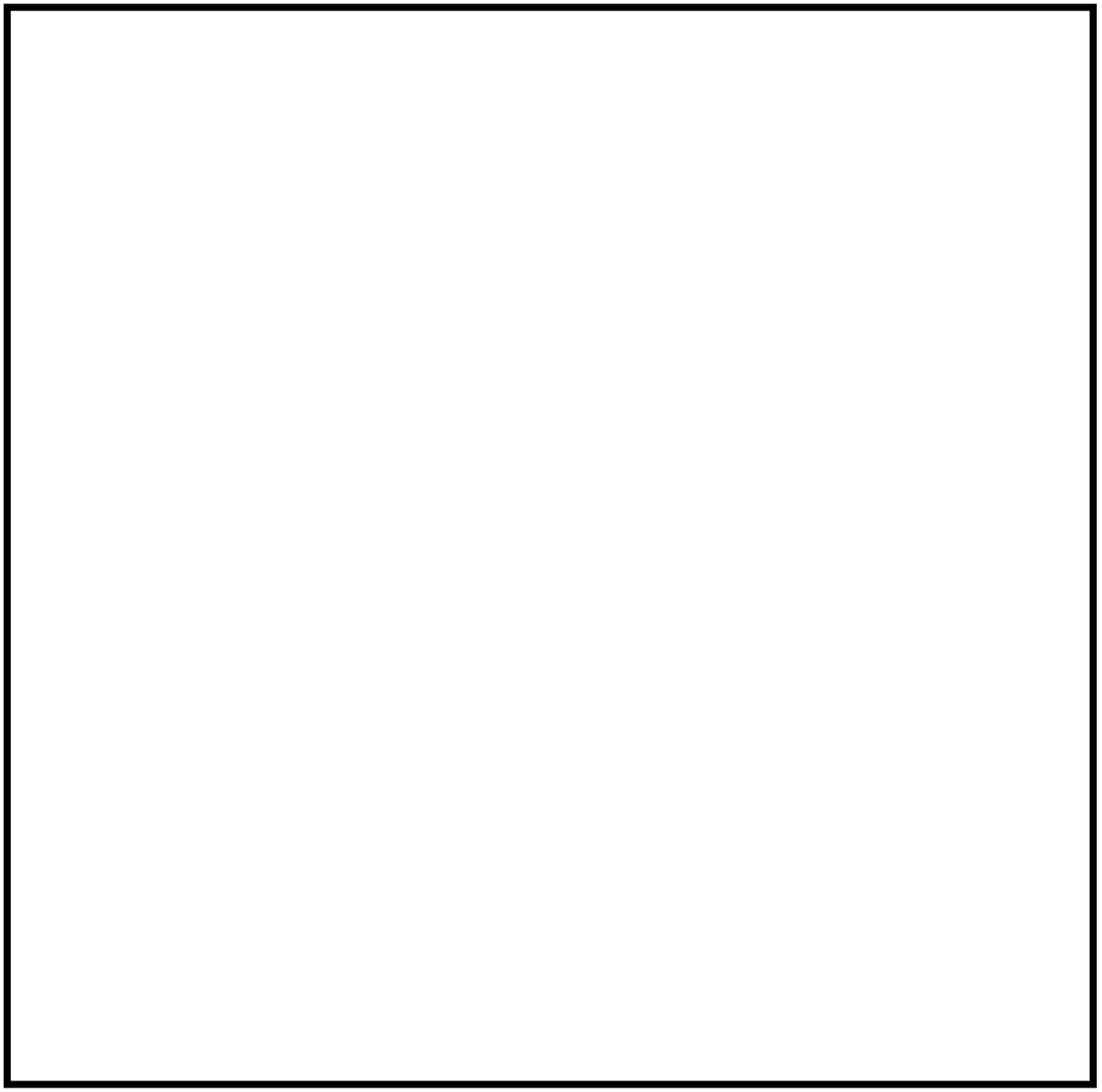
```
System.out.println( … )
System.out.print( … )
```

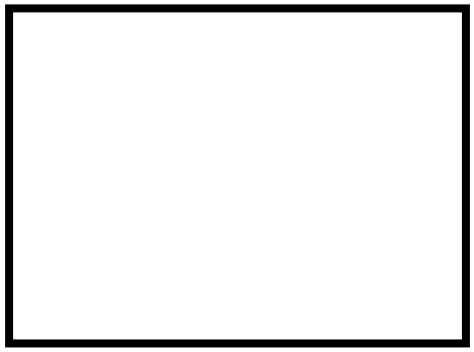
Méthodes/attributs de la classe Math

```
Math.sin(rad) //cos, tan, asin,
              // acos, PI
Math.exp(x) // aussi log(x)
Math.log(x) // aussi log10(x)
Math.pow(x, a) // x à la
               // puissance a
Math.sqrt(x)
Math.random()
Math.abs(x)
```

5. Section 1 - Exercices

Mémoire de programme **Exercice 1-1 -** Algorithme pour moyenne **Mémoire de travail**





UCT

Mémoire de programme **Exercice 1-2 – Autre exemple** **Mémoire de travail**

DONNÉES:
RÉSULTATS:
EN-TÊTE:
MODULE:

DONNÉES:
INTERMÉDIAIRES:
RÉSULTATS:
EN-TÊTE:
MODULE:

UCT

Mémoire de programme **Exercice 1-3 Moyenne sur 100** **Mémoire de travail**

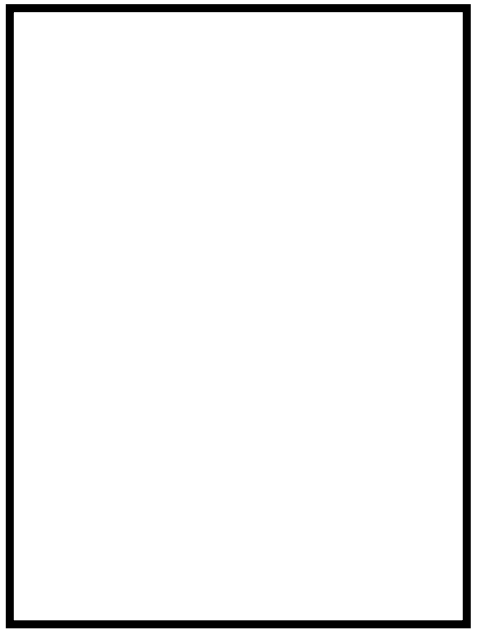
DONNÉES:

RÉSULTATS:

INTERMÉDIAIRES:

EN-TÊTE:

MODULE:



UCT

Mémoire de programme **Exercice 1-4 Dernier Exemple** **Mémoire de travail**

… Sans constantes

…

MODULE

…

T1 ← C1 * 0.07 // GST

T2 ← C2 * 0.07 // GST

T3 ← C3 * 0.07 // PST

T4 ← C4 * 0.07 // GST

…

…avec constantes TPS et TVQ

MODULE

UCT

6. Section 2 - Exercices

Exercice 2-1 Ordre des caractères:

`'A' < 'a'` est

alors que

`'?' < ' '` est

Exercice 2-2 - Test pour majuscule

- Soit la variable **x** contenant une valeur de type **char**.
- Écrivez une expression Booléenne qui est VRAIE si la valeur de **x** est une lettre majuscule et FAUSSE sinon.
  - Notez que nous n'avons pas besoin de connaître les valeurs codées de ces caractères!

Exercice 2-3 - Précédence des opérateurs

- Quelle est l'ordre d'évaluation des expressions suivantes?

```
a + b + c + d + e
```

```
a + b * c - d / e
```

```
a / (b + c) - d % e
```

```
a / (b * (c + (d - e)))
```

Mémoire de programme **Exercice 2-4 - Un peu de math** **Mémoire de travail**

```
// Déclarations de variables



// Calcul de la moyenne

// Déclarations de variables



// Calcule quotient et restant
```

UCT

7. Section 3 – Exercices

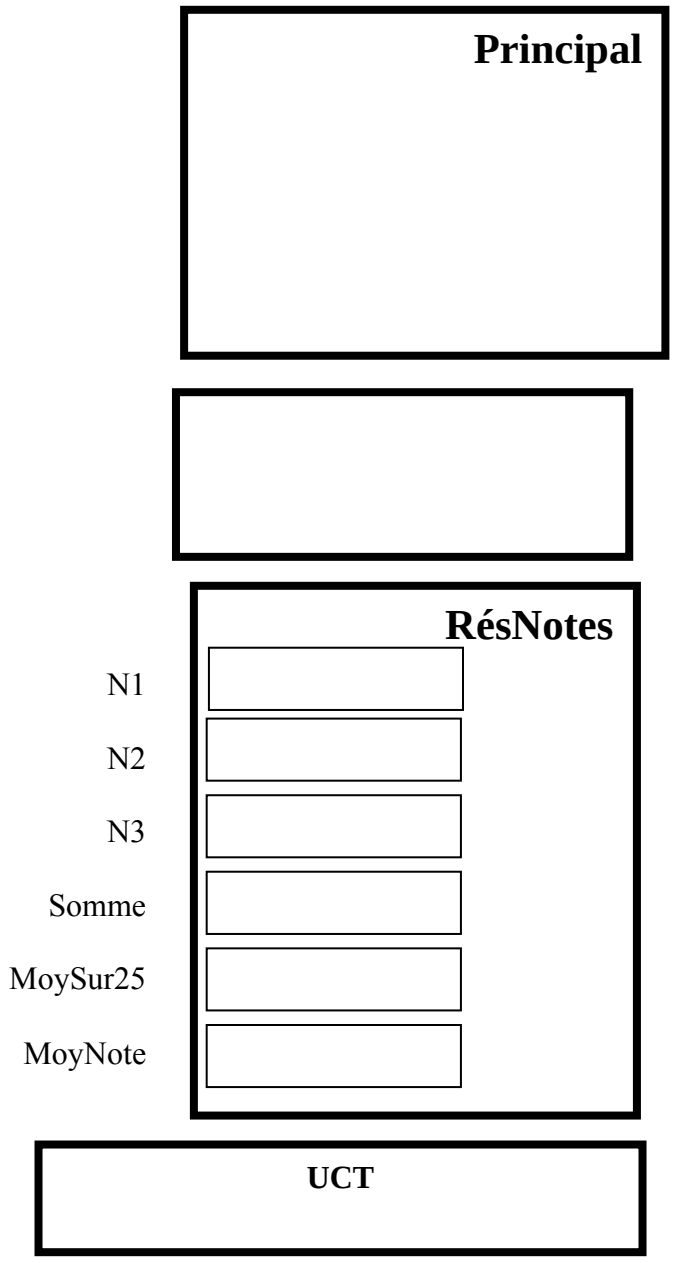
Mémoire de programme | **Exercice 3-1 – Algorithme principal** | **Mémoire de travail**

DONNÉES:
RÉSULTATS:
INTERMÉDIAIRES:

EN-TÊTE:
MODULE:

DONNÉES: **N1, N2, N3 (notes sur 25)**
RÉSULTATS: **MoyNotes (moyenne de N1, N2, N3 sur 100)**
INTERMÉDIAIRES: **Somme (somme des notes)**
MoySur25 (moyenne des notes, sur 25)
EN-TÊTE: **MoyNote ← RésNotes(N1, N2, N3)**
MODULE:

1. **Somme ← N1 + N2 + N3**
2. **MoySur25← Somme / 3**
3. **MoyNote ← MoySur25 * 4**



Mémoire de programme

Exercice 3-2 – Échanger deux valeurs

Mémoire de travail

DONNÉES:
RÉSULTATS:
MODIFIÉS:
INTERMÉDIAIRES:

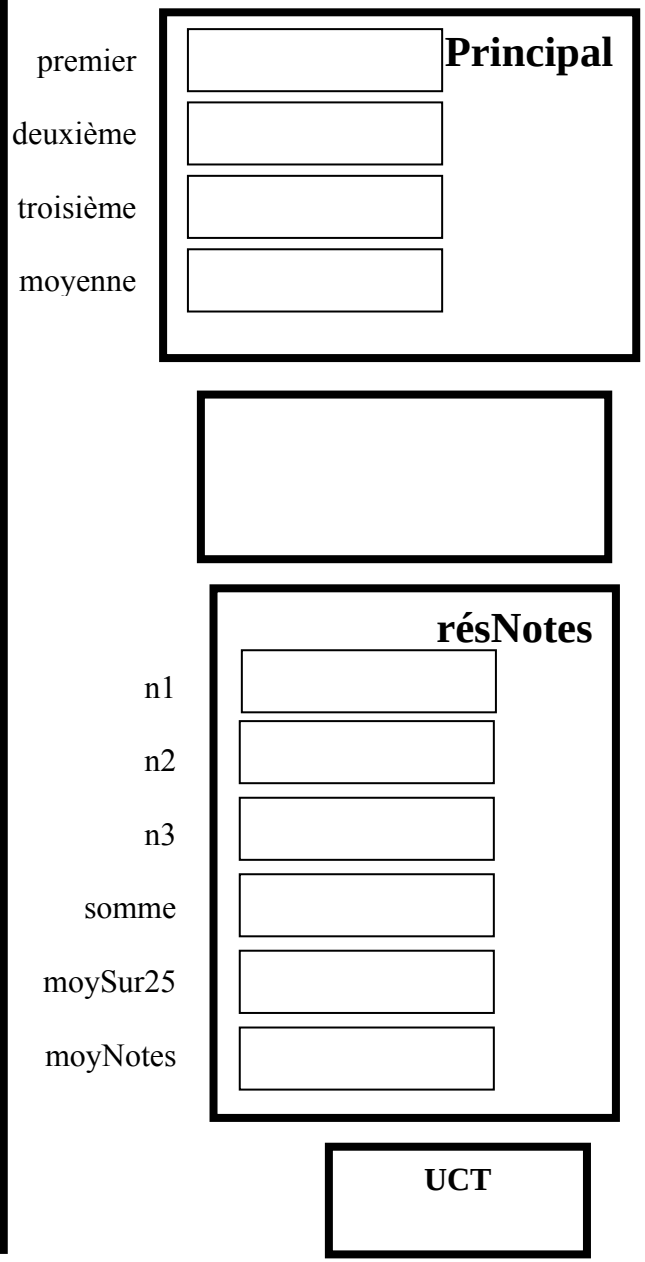
EN-TÊTE:

MODULE:

UCT

Mémoire de programme **Exercice 3-3 – Traduction au Java** **Mémoire de travail**

```
 public static void main (String[] args)
 {
    // MISE EN PLACE DE LA LECTURE AU CLAVIER
    Scanner clavier = new Scanner( System.in );
    // DÉCLARATIONS DES VARIABLES ET DICTION. DE DONNÉES


    // LECTURE DES VALEURS DE L'UTILISATEUR




    // INVOCATION DE resMoy

    // AFFICHAGE DES RÉSUTATS ET MODIFIÉES À L'ÉCRAN

}
 public static double résNotes(double n1, double n2,
                                double n3)
 {
    //DÉCLARATIONS DES VARIABLES ET DICTION. DE DONNÉES
    //INTERMÉDIAIRES


    // RÉSULTAT

    // MODULE DE L'ALGORITHME


    // RETOURNER RÉSULTAT
    return(               );
}
```



8. Section 4 - Exercises

Mémoire de programme **Exercice 4-1 Exemple de traçage** **Mémoire de travail**

Invocation: MoyNotes ← RésNotes(18, 23, 19)

DONNÉES: N1, N2, N3 (notes sur 25)
RÉSULTATS: MoyNotes (moyenne des notes, sur 100)
INTERMÉDIAIRES:
Somme (somme des notes)
MoySur25 (moyenne des notes, sur 25)
EN-TÊTE: MoyNotes ← RésNotes(N1, N2, N3)
MODULE:
1. Somme ← N1 + N2 + N3
2. MoySur25 ← Somme / 3
3. MoyNotes ← MoySur25 * 4

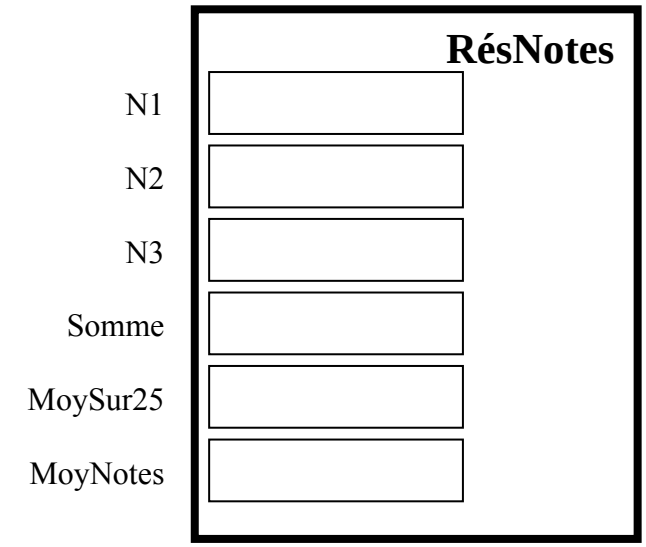


Tableau de traçage pour MoyNotes ← RésNotes(18, 23, 19)

Instruction	N1	N2	N3	Somme	MoySur25	MoyNotes
Valeurs initiales						

Mémoire de programme **Exercice 4-2 Tracer une invocation** **Mémoire de travail**

DONNÉES: (aucune)
RÉSULTATS: (aucune)
INTERMÉDIAIRES:
  Premier, Deuxième, Troisième (trois notes)
  Moyenne (Moyenne des notes, sur 100)
EN-TÊTE: Principal()
MODULE:
  (Lires les notes de l'utilisateur)
1. AfficheLigne("S.V.P. tapez trois résultats sur 25")
2. Premier ← LireRéel()
3. Deuxième ← LireRéel ()
4. Troisième ← LireRéel ()
  (Invocation de l'algorithme RésNotes)
5. Moyenne ← RésNotes(Premier, Deuxième, Troisième)
  (Affiche la moyenne pour l'utilisateur)
6. AfficheLigne("La moyenne est ", Moyenne)

DONNÉES: N1, N2, N3 (notes sur 25)
RÉSULTATS: MoyNotes (Moyenne des notes, sur 100)
INTERMÉDIAIRES: Somme (Somme des notes)
  MoySur25 (Moyenne des notes, sur 25)
EN-TÊTE: MoyNotes ← RésNotes(N1, N2, N3)
MODULE:
1. Somme ← N1 + N2 + N3
2. MoySur25 ← Somme / 3
3. MoyNotes ← MoySur25 * 4

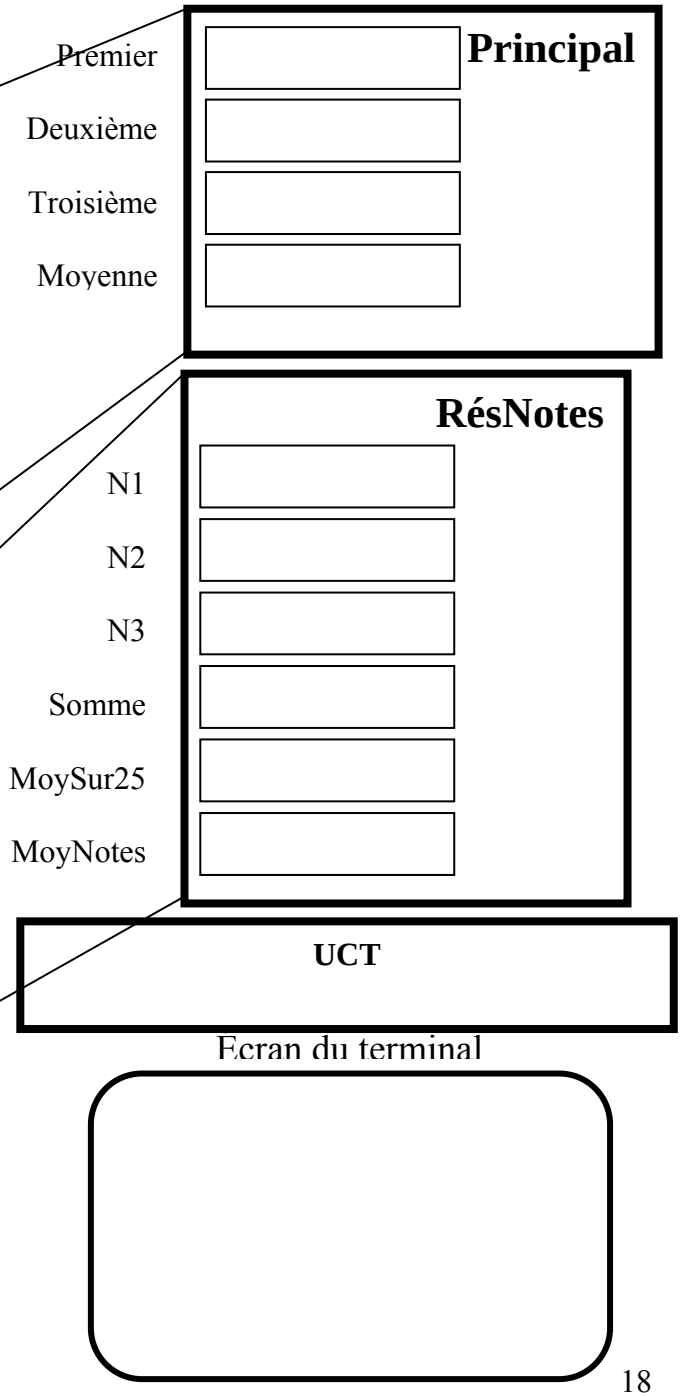


Table 1 – Traçage pour l'algorithme Principal:

Interaction avec l'utilisateur:

S.V.P. tapez trois résultats sur 25

23 16 21

La moyenne est 80

Instructions	Premier	Deuxième	Troisième	Moyenne
Valeur initiales	**?**	**?**	**?**	**?**

Invocation algorithme RésNotes:

Moyenne ← RésNotes (Premier, Deuxième, Troisième)

MoyNotes ← RésNotes (N1, N2, N3)

Table 2 – Traçage pour MoyNotes ← RésNotes(23,16,21)

Instructions	N1	N2	N3	Somme	MoySur25	MoyNotes
Valeur initiales						

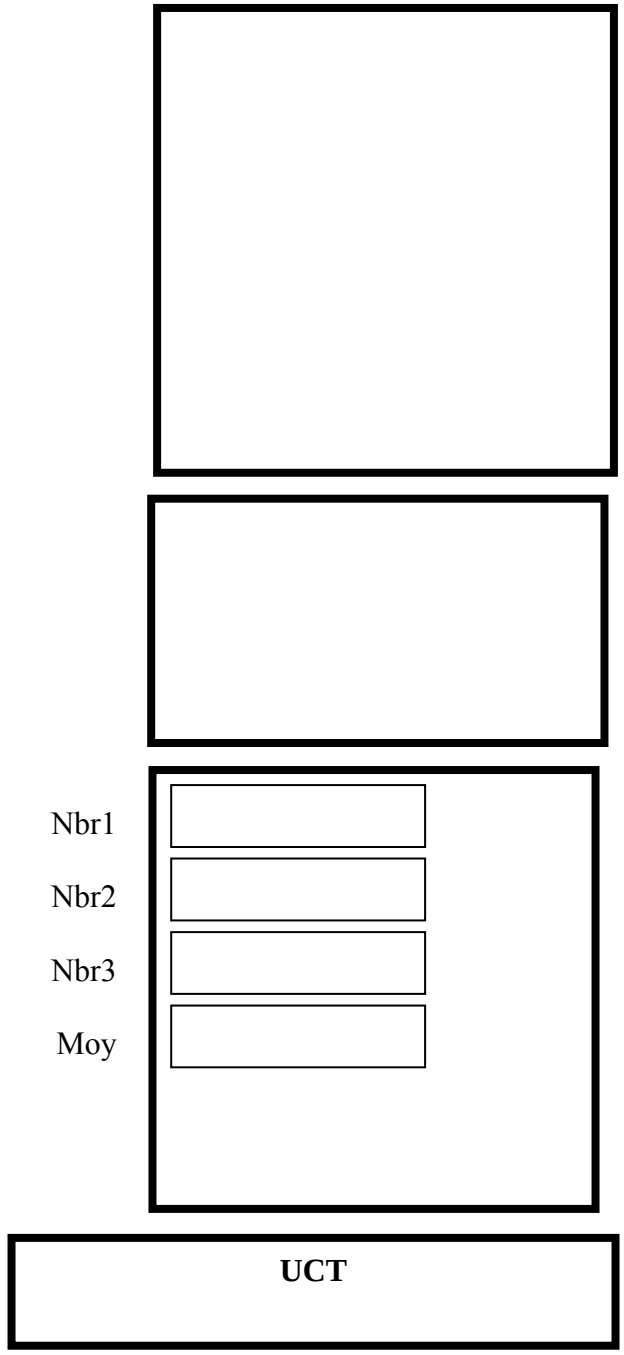
Mémoire de programme **Exercice 4-3** – Problème de moyenne à nouveau! **Mémoire de travail**

DONNÉES:
RÉSULTATS:
INTERMÉDIAIRES:

EN-TÊTE:
MODULE:

DONNÉES: Nbr1, Nbr2, Nbr3 (trois nombres)
RÉSULTATS: Moy (la Moyenne de Nbr1, Nbr2, et Nbr3)
EN-TÊTE: Moy ← Moyenne(Nbr1, Nbr2, Nbr3)
MODULE:

1. Moy ← (Nbr1 + Nbr2 + Nbr3)/3



Mémoire de programme **Exercice 4-4 – Exemple de traçage** **Mémoire de travail**

Invocation de Moyenne ← RésNotes(23, 16, 21)

DONNÉES: N1, N2, N3 (notes sur 25)
RÉSULTATS: MoyNotes (Moyenne des notes, sur 100)
INTERMÉDIAIRES:
MoySur25 (Moyenne des notes, sur 25)
EN-TÊTE: MoyNotes ← RésNotes(N1, N2, N3)
MODULE:
1. MoySur25 ← Moyenne(N1, N2, N3)
2. MoyNotes ← MoySur25 * 4

DONNÉES: Nbr1, Nbr2, Nbr3 (trois nombres)
RÉSULTATS: Moy (la Moyenne of Nbr1, Nbr2, et Nbr3)
EN-TÊTE: Moy ← Moyenne(Nbr1, Nbr2, Nbr3)
MODULE:
1. Moy ← (Nbr1 + Nbr2 + Nbr3)/3

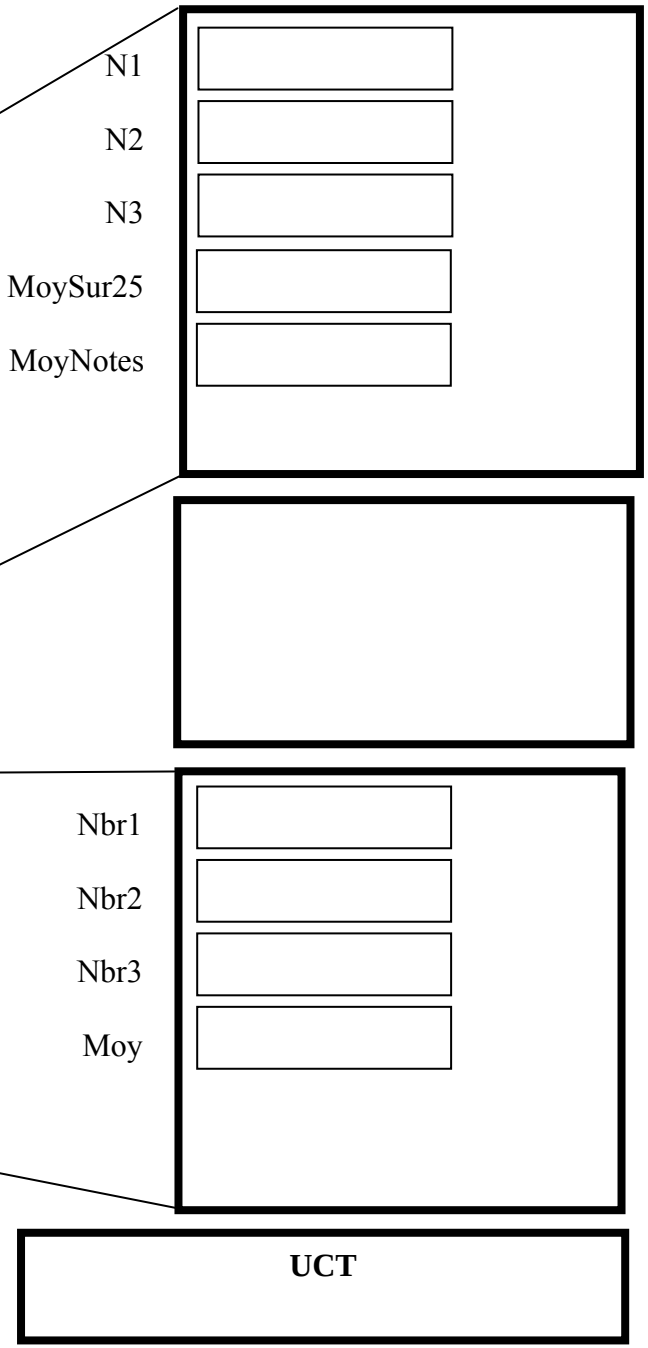


Table 1 – Traçage pour MoyNotes ← RésNotes(23, 16, 21)

Instructions	N1	N2	N3	MoySur25	MoyNotes
Valeur initiales					

Invocation de l'algorithme Moyenne:
MoySur25 ← Moyenne (N1, N2, N3)

Moy ← Moyenne (Nbr1, Nbr2, Nbr3)

Table 2 - Traçage pour Moy ← Moyenne(23, 16, 21)

Instructions	Nbr1	Nbr2	Nbr3	Moy
Valeur initiales				
1. Somme ← (Nbr1 + Nbr2 + Nbr3)/3				

Mémoire de programme

Exercice 4-5 – Chiffres inversés

Mémoire de travail

DONNÉES:

RÉSULTATS:

IINTERMÉDIAIRES:

EN-TÊTE:

MODULE:

L'algorithme suivant est disponible pour extraire les chiffres (dizaines et unités) d'un nombre:

(Premier, Second) ← Chiffres(X)

UCT

Table 1 - Traçage pour Ninversé ← Inverse2Chiffres(42))

Instructions	N	Diz	Unités	Ninversé
Valeur initiales				
1. Invocation Chiffres(N)				
2. Ninversé ← Unités * 10 + Diz				

Invocation de (Diz, Unités) ← Chiffres (N)

(Diz, Unités) ← Chiffres (N)

4 2 42

(Premier, Second) ← Chiffres (X)

Mémoire de programme **Exercice 4-7 – Joindre 4 nombres** **Mémoire de travail**

DONNÉES:

RÉSULTAT:

INTERMÉDIAIRES:

EN-TÊTE

MODULE:

Vous pouvez supposer que vous disposez de l'algorithme:

C ← Joindre(A, B)

DONNÉES: A, B (deux entiers positifs)
RÉSULTATS: C (le nombre constitué des chiffres de A suivis des chiffres de B)

UCT

9. Section 5 Exercices

Mémoire de programme **Exercice 5-1 - De retour au problème du plus grand nombre** **Mémoire de travail**

DONNÉES:
RÉSULTATS:
EN-TÊTE:
MODULE:

UCT

Mémoire de programme **Exercice 5-2 - Maximum entre 3 nombres** **Mémoire de travail**

DONNÉES:
RÉSULTATS:
EN-TÊTE:
MODULE:
(Branchements imbriqués)

OU
(Séquences de branchements)

UCT

DONNÉES: X, Y, Z (trois nombres)
RÉSULTATS: M (la plus grande des données)
EN-TÊTE: M ← Max3(X, Y, Z)
MODULE:

Séquences de branchements

```
public double max3(double x,
                   double y, double z)
{




}
```

UCT

DONNÉES: X, Y, Z (trois nombres)
RÉSULTATS: M (la plus grande des données)
EN-TÊTE: M ← Max3(X, Y, Z)
MODULE:

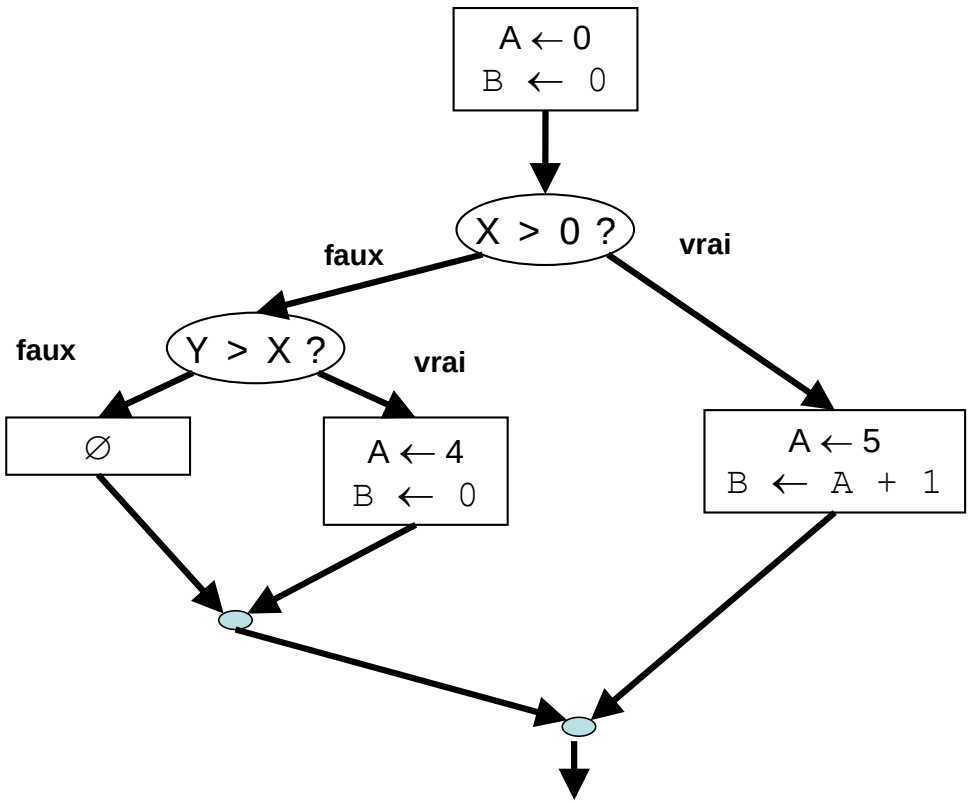
Branchements imbriqués

```
public double max3(double x,
                   double y, double z)
{




}
```

UCT

Mémoire de programme **Exercice 5-5 – Traduction d'un bloc d'instruction** **Mémoire de travail**

{

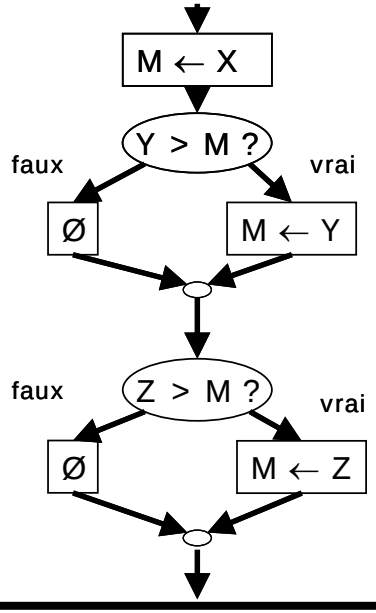


}

UCT

Mémoire de programme **Exercice 5-5 – Trace de Max3 version 2** **Mémoire de travail**

DONNÉES: X, Y, Z (trois nombres)
RÉSULTATS: M (la plus grande des données)
EN-TÊTE: M ← Max3(X, Y, Z)
MODULE:



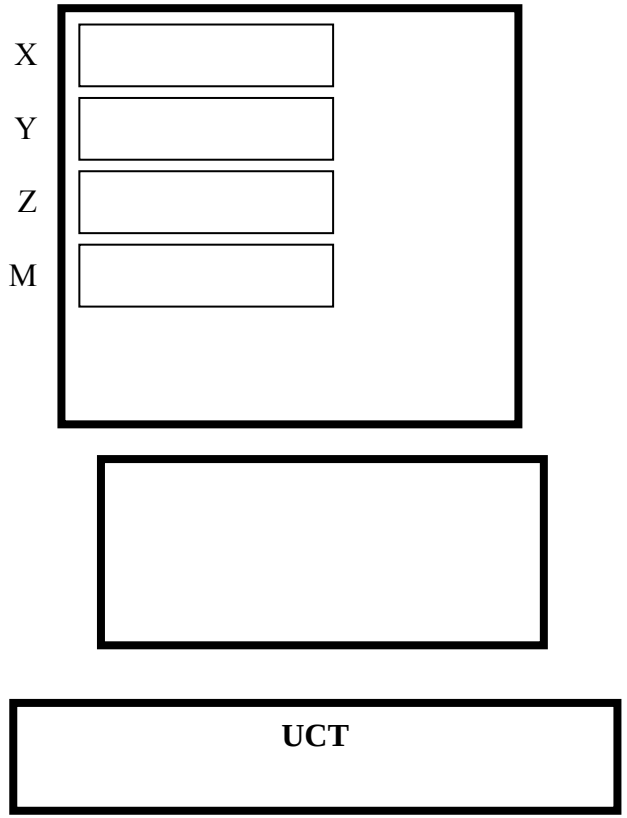


Table 1 – Traçage pour M ← Max3(5, 11, 8)

	X	Y	Z	M
Valeurs initiales				

DONNÉES: Âge (âge de la personne)
RÉSULTATS: Coût (coût du billet)
EN-TÊTE: Coût ← CoûtBillet(Âge)
MODULE:

(Version 1: Branchements imbriqués)

(Version 2: Séquences de branchements)

UCT

Mémoire de programme **Exercice 5-8 – Valeur positive** **Mémoire de travail**

DONNÉE:
RÉSULTAT:

EN-TÊTE:
MODULE

UCT

Exercice 5-9 Expressions Booléennes composées

Exercice 5-10 - Encore des expressions Booléennes complexes

Supposons que X = 5 et que Y = 10.

Expression	Valeur
(X > 0) ET (NON (Y = 0))	
(X > 0) ET ((X < Y) OU (Y = 0))	
(NON (X > 0)) OU ((X < Y) ET (Y = 0))	
NON ((X > 0) OU ((X < Y) ET (Y = 0)))	

10. Section 6 - Exercices

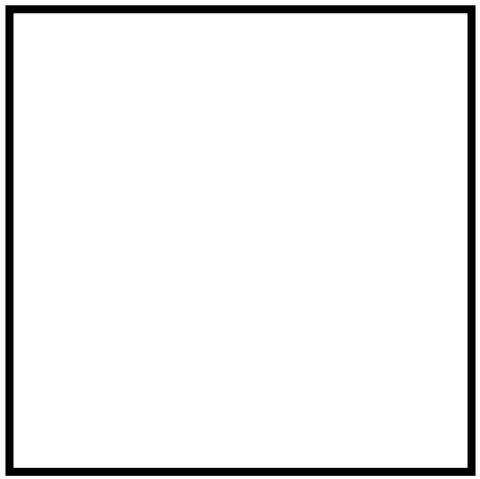
Mémoire de programme | **Exercice 6-1 - Somme de 1 à N** | **Mémoire de travail**

DONNÉES:
INTERMÉDIAIRES:
RÉSULTATS:
EN-TÊTE:
MODULE:

UCT

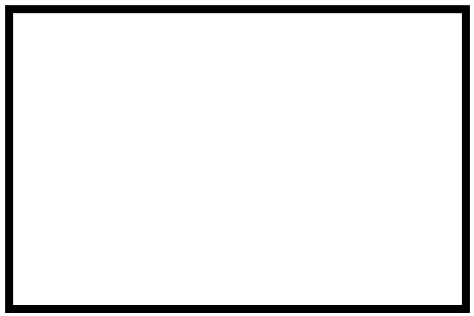
Exercice 6-1: Trace de Somme1àN(3)

Mémoire de travail



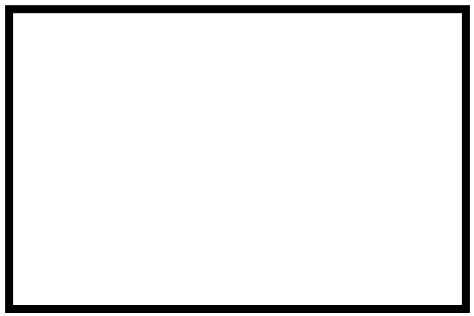
	N	Compteur	Somme
Init.			

DONNÉES:
INTERMÉDIAIRES:
RÉSULTATS:
EN-TÊTE:
MODULE:



UCT

DONNÉES:
INTERMÉDIAIRES:
RÉSULTATS:
EN-TÊTE:
MODULE:



UCT

Mémoire de programme

Exercice 6-2 – Boucle déterminée

Mémoire de travail

DONNÉES:
RÉSULTATS:
INTERMÉDIAIRES:
HYPOTHÈSES:
EN-TÊTE:
MODULE:

UCT

Mémoire de programme **Exercice 6-3 – Boucle indéterminée** **Mémoire de travail**

DONNÉES:

RÉSULTATS:

INTERMÉDIAIRES:

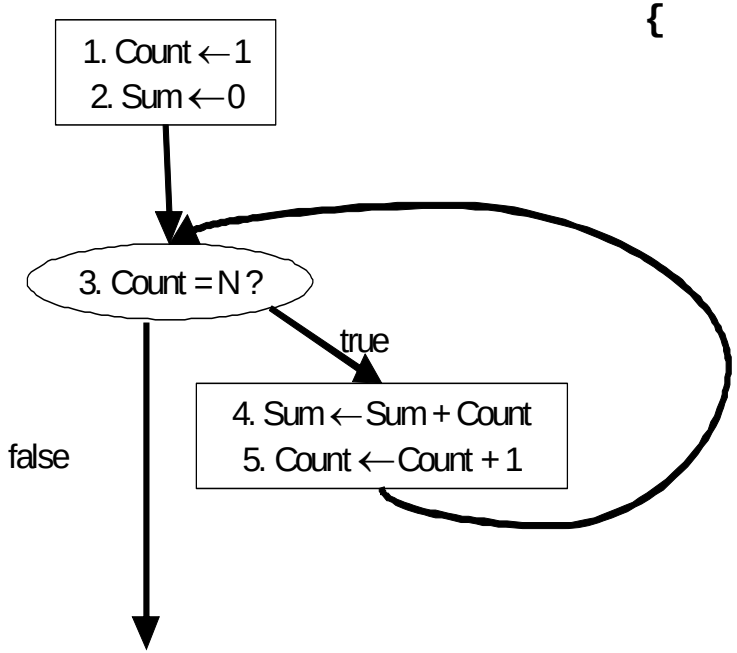
HYPOTHÈSES:

EN-TÊTE:

MODULE:

UCT

DONNÉES: N (un entier positif)
INTERMÉDIAIRES: Compteur (compteur de 1 à N)
RÉSULTATS: Somme (somme des entiers de 1 à N)
EN-TÊTE: Somme ← Somme1àN(N)
MODULE:




```
public static int somme1àN (int n)
{
}
```

UCT

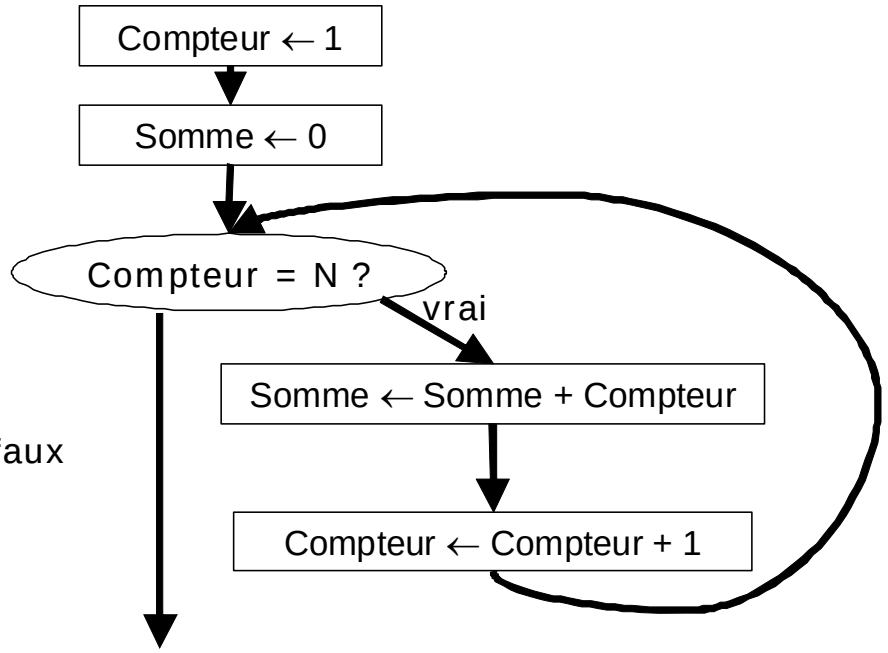
DONNÉES:
RÉSULTATS:
INTERMÉDIAIRES:

CONSTRAINTS:
EN-TÊTE: **Principal()**
MODULE:

```
public static void main(String args)
{




}
```

UCT

Mémoire de programme **Exercice 6-6 – Somme1àN avec boucle FOR** **Mémoire de travail**




```
// Traduction au Java
```

UCT

Exercice 6-7 – Utilisation des indices

- L'indice d'un tableau T de taille L peut être n'importe quelle expression mathématique retournant un entier entre 0 et L-1.
  - Soit K =2, et T réfère à

T[2] =
T[K] =
T[2*K-1] =
T[T[0]+1] =

- T[expression] peut être utilisé comme toute autre variable dans des expressions plus complexes.
- **Rappelez-vous T est une variable de référence**

Mémoire de programme **Exercice 6-8** – Valeur au milieu d'un tableau **Mémoire de travail** **Mémoire globale**

DONNÉES: T (réfère à un tableau de nombres réels)
N (un entier impair, la longueur du tableau)
RÉSULTATS: ValMil (nombre au milieu du tableau
INTERMÉDIAIRES:
EN-TÊTE:
MODULE

UCT

Mémoire de programme **Exercice 6-9 – Échanger les valeurs d'un tableau** **Mémoire de travail** **Mémoire globale**

DONNÉES: T (réfère à un tableau d'entiers)
I, J (deux indices)
MODIFIEDS:
INTERMÉDIAIRES:

RÉSULTATS:
EN-TÊTE: Échange(T, I, J)
MODULE

UCT

Mémoire de programme **Exercice 6-10 – Création d'un nouveau tableau** **Mémoire de travail** **Mémoire globale**

DONNÉES: N (un entier positif)
RÉSULTATS: T (une reference à un tableau contenant N…1)
INTERMÉDIAIRES:
EN-TÊTE:
MODULE:

Trace pour N=3

Instructions	N	Index	T
Valeurs initiales	**3**	**?**	

<table>
<tr><th>Modèle algorithmique</th><th>Java</th></tr>
<tr><td colspan="2">Exercice 6-11 – Somme des valeurs d'un tableau</td></tr>
<tr><td>DONNÉES: A (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
INTERMÉDIAIRES:
RÉSULTATS: Somme (Somme du contenu du tableau)
EN-TÊTE: Somme ← SommeTableau(T,N)
MODULE:</td><td><code>public static int[] sommeTableau(int[] t)</code>
<code>{</code>

<code>}</code></td></tr>
<tr><td colspan="2">Exercice 6-12 (a) – Soit une valeur V et un tableau X de N valeurs, vérifiez si la somme des valeurs de X excède V.</td></tr>
<tr><td>DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
V (Une valeur limite)
INTERMÉDIAIRES:

RÉSULTATS: Excède (Booléen: Vrai si Somme > V
sinon faux)
EN-TÊTE: Excède ← SommeExcèdeT(T,N,V)
MODULE:</td><td><code>public static boolean sommeExcède(int[] t, int v)</code>
<code>{</code>

<code>}</code></td></tr>
</table>

<table>
<tr><th>Modèle algorithmique</th><th>Java</th></tr>
<tr><td colspan="2">Exercice 6-12 (b) – Soit une valeur V et un tableau X de N valeurs, vérifiez si la somme des valeurs de X excède V.</td></tr>
<tr><td>DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
V (Une valeur limite)
INTERMÉDIAIRES:

RÉSULTATS: Excède (Booléen: Vrai si Somme > V sinon faux)
EN-TÊTE: Excède ← SommeExcèdeT(T,N,V)
MODULE:</td><td><code>public static boolean sommeExcèdeT (int[] t, int v)</code>
<code>{</code>

<code>}</code></td></tr>
<tr><td colspan="2">Exercice 6-13 – Comptez combien de fois K apparaît dans un tableau de taille N.</td></tr>
<tr><td>DONNÉES: A (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
K (valeur dont on va compter les instances)
INTERMÉDIAIRES:
Index (Index de tableau de 0 à N-1)
RÉSULTATS: Compteur (Nombres d'intances de K dans T
EN-TÊTE: Compteur ← CompteK(T,N,K)
MODULE:</td><td><code>public static int compteK(int[] t, int k)</code>
<code>{</code>

<code>}</code></td></tr>
</table>

Modèle algorithmique	Java
Exercice 6-14 (a) – Soit un tableau T de N valeurs et un nombre K, vérifiez si K apparaît dans T ou non.	
DONNÉES: T (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) K (Valeur recherchée dans tableau) INTERMÉDIAIRES: RÉSULTATS: Trouvé (Booléen: vrai si K est dans tableau, sinon faux EN-TÊTE: Trouvé ← TrouveK(T,N,K) MODULE: Compteur ← CompteK(T,N,K) Trouvé ← Compteur > 0	`public static boolean trouveK(int[] t, int k)` `{` `}`
Exercice 6-14 (b) – Soit un tableau T de N valeurs et un nombre K, vérifiez si K apparaît dans T ou non.	
DONNÉES: T (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) K (Valeur recherchée dans tableau) INTERMÉDIAIRES: RÉSULTATS: Trouvé (Booléen: vrai si K est dans tableau, sinon faux) EN-TÊTE: Trouvé ← TrouveK(T,N,K) MODULE:	`public static boolean trouveK(int[] t, int k)` `{` `}`

Modèle algorithmique	**Java**
Exercice 6-15 – Soit un tableau X de N valeurs et un nombre K, trouvez la position de la première occurrence de K. (Si K n'est pas dans X, retournez –1 comme position.)	
DONNÉES: A (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) K (Valeur recherchée dans tableau) INTERMÉDIAIRES: RÉSULTATS: Position (Position de K dans tableau, ou -1 si K n'est pas dans le tableau) EN-TÊTE: Position ← OùEstK(T,N,K) MODULE:	`public static int oùEstK(int t[], int n, int k)` `{` `}`

Modèle algorithmique	Java
Exercice 6-16 – Trouvez la valeur maximale dans un tableau de taille N.	
DONNÉES: T (réfère à un tableau de N valeurs) N (nombres de valeurs dans le tableau) INTERMÉDIAIRES : RÉSULTATS: Max (valeur maximale dans le tableau) EN-TÊTE: Max ← MaxDansTableau(T, N) MODULE	`public static int maxDansTableau(int [] t)` `{` `}`

Modèle algorithmique	Java
Exercice 6-17 (a) - Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.	
DONNÉES: T (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) INTERMÉDIAIRES: RÉSULTATS: Position (Position de la première valeur maximale dans le tableau) EN-TÊTE: Position ← MaxPosDansTableau(T,N) MODULE:	`public static int maxPosDansTableau(int [] t)` `{` `}`
Exercice 6-17 (b) – Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.	
DONNÉES: T (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) INTERMÉDIAIRES: RÉSULTATS: Position (Position de la première valeur maximale dans le tableau) EN-TÊTE: Position ← MaxPosDansTableau(T,N) MODULE:	`public static int maxPosDansTableau(int [] t)` `{` `}`

Modèle algorithmique	Java
Exercice 6-17 (c) – Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.	
DONNÉES: T (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) INTERMÉDIAIRES: RÉSULTATS: Position (Position de la première valeur maximale dans le tableau) EN-TÊTE: Position ← MaxPosDansTableau(T,N) MODULE:	`public static int maxPosDansTableau(int [] t)` `{` `}`

Modèle algorithmique	**Java**
Exercice 6-18 – Vérifiez si un tableau de N valeurs contient ou non des valeurs dupliquées.	
DONNÉES: T (Réfère à un tableau de nombres) N (Nombre d'éléments dans le tableau) INTERMÉDIAIRES: RÉSULTATS: Doubles (Booléen: vrai si tableau a des doubles, sinon faux) EN-TÊTE: Doubles ← AdesDoubles(T,N) MODULE:	`public static boolean aDesDoubles(int [] t)` `{` `}`

Exercice 6-19: Comparaisons de chaînes **`String`**

- Quelle est la valeur de **`résultat`** pour ces exemples?
  - Exemple 1:

```
String str1 = "abcde" ;
String str2 = "abcfg" ;
int résultat = str1.compareTo(str2);
```

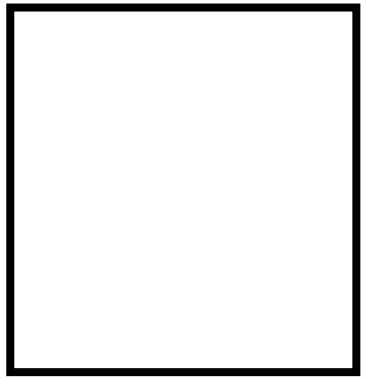
  - Exemple 2:

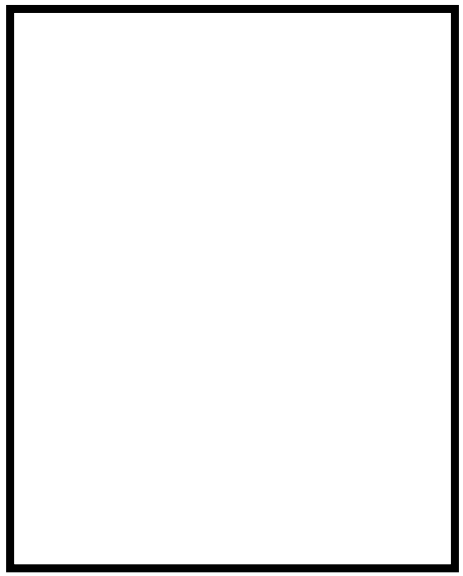
```
String str1 = "abcde" ;
String str2 = "ab" ;
int résultat = str1.compareTo(str2);
```

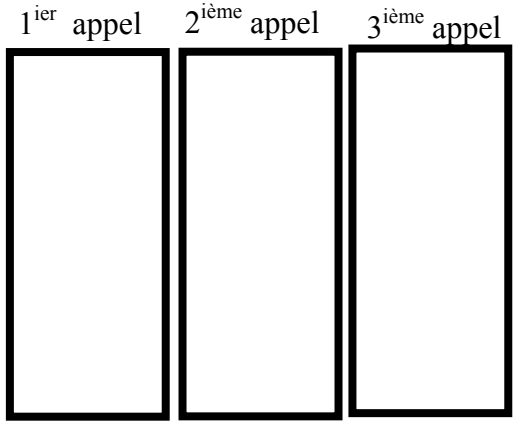
11. Section 7 - Exercices

Mémoire de programme **Exercice 7-1 – Échanges de valeurs dans tableau** **Mémoire de travail** **Mémoire globale**

```
class ÉchangeComplet
{
  public static void main (String args[ ])
  { int i = 0;
    int[ ] t;
    t = new int[]
                { 2, 4, 6, 8, 10, 12 } ;
    while( i <= 2 )
    {
      échangeTbl(t, i, 5 - i ) ;
      i = i + 1;
    }
    for ( i = 0 ; i <= 5 ; i = i + 1 )
    { System.out.println("t[" + i +
      "] vaut " + t[i] );
    }
  }
  // échangeTbl : échanges valeurs à i,j
  // Données: x, reference à un tableau,
  //        i,j, 2 indices de x
  public static void échangeTbl(
                             int[ ] x,
                             int i,int j)
   {
      // DÉCLARE VARIABLES
      int temp ; // Inter: contient x[i]
      // MODULE DE L'ALGORITHME
      temp = x[i] ;
      x[i] = x[j] ;
      x[j] = temp;
```







Écran de terminal

UCT

Exercice 7-1: Trace – Table 1, page 1, main

Instructions	i	t	Tableau	Sortie
Valeurs initiales	?	?	?	

Instructions	i	t	Tableau	Sortie
(Valeurs les plus récentes de la page 1)				

Exercice 7-1: Trace – Table 2, échangeTbl(a, 0, 5)

```
échangeTbl( t, i, 5-i)
échangeTbl( x, i,  j )
```

Instructions	x	i	j	temp	Tableau de Table 1
Valeurs initiales		0	5		
				2	

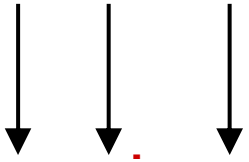
Exercice 7-1: Trace – Table 3, échangeTbl(a, 1, 4)

```
échangeTbl( t, i, 5-i)
échangeTbl( x, i,  j )
```

Instructions	x	i	j	temp	Tableau de Table 1
Valeurs initiales					
				4	

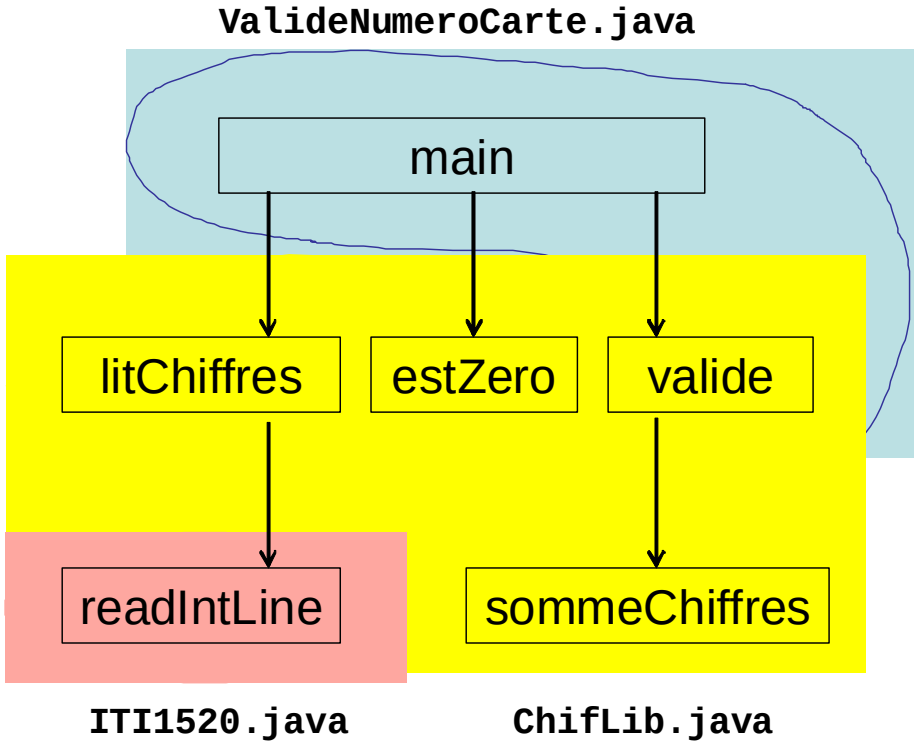
Exercice 7-1: Trace – Table 4, échangeTbl(a, 2, 3)

```
échangeTbl( t, i, 5-i)
```



```
échangeTbl( x, i,  j )
```

Instructions	x	i	j	temp	Tableau de Table 1
Valeurs initiales					


ValideNumeroCarte.java
main
litChiffres
estZero
valide
readIntLine
sommeChiffres
ITI1520.java
ChifLib.java

Modèle algorithmique	Java
Classe ValideNuméroCarte (ValideNuméroCarte.java)	
DONNÉES: (aucune) RÉSULTAT: (aucune) INTERMÉDIAIRES: chiffres (référence à un tableau d'entiers) n (nombre d'éléments dans le tableau) testValide (Indique si la carte est valide) EN-TÊTE: main MODULE:	`/* La méthode main, tel un chef d'orchestre, invoque les autres méthodes afin de compléter les tâches individuelles. */` `public static void main (String [ ] args)` `{` `  // invoque litChiffres( ) pour saisir la donnée` `  int [ ] chiffres = ChifLib.litChiffres( );` `  while ((chiffres.length == 4) && (!ChifLib.estZero(chiffres)))` `  {` `  // envoie ce nombre à la méthode valide` `  boolean testValide = valide(chiffres);` `  // affiche le résultat` `  if (testValide)` `  { System.out.println("Ce numéro est valide."); }` `  else` `  { System.out.println("Ce numéro est invalide."); }` `  }` `}`
DONNÉES: chiffres (référence à un tableau de chiffres) n (nombre d'éléments dans le tableau) RÉSULTAT: estValide (VRAI si carte est valide, sinon FAUX) INTERMÉDIAIRES: troisPremiers (trois premiers chiffres du dernier groupe) dernierChiffre (tout dernier chiffre du numéro) somme (somme des 15 chiffres) EN-TÊTE: estValide ← valide(chiffres, n) MODULE:	`/* Cette méthode invoque sommeChiffres( ) pour trouver la somme des chiffres d'un entier. Elle compare ensuite les derniers chiffres de la somme des 15 premiers chiffres et du numéro lui-même pour déterminer la validité de la carte. NOTE: Peut être privée!!! */` `private static boolean valide(int [ ] chiffres)` `{` `  // trouve les 3 premiers chiffres du dernier groupe` `  int troisPremiers =` `  // trouve le tout dernier chiffre du numéro` `  int dernierChiffre =` `  // trouve la somme des 15 premiers chiffres` `  int somme =` `  // détermine la valididé` `  boolean valide =` `  return valide;` `}`

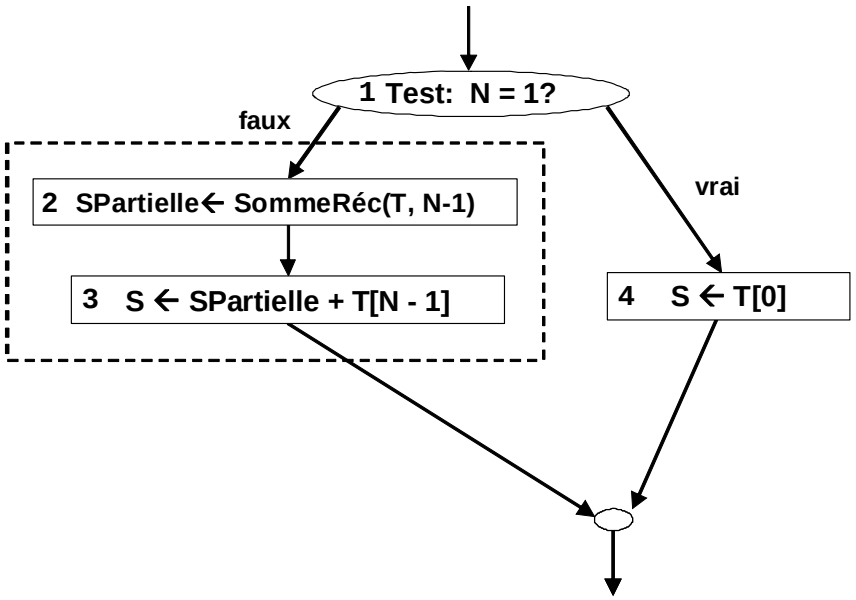
Modèle algorithmique	Java
Classe ChifLib (ChifLib.java)	
DONNÉES: chiffres (référence à un tableau d'entiers) n (nombre d'éléments dans le tableau) RÉSULTAT: drapeau (VRAI si le premier chiffre est zéro, sinon FAUX) INTERMÉDIAIRES: (aucune) EN-TÊTE: drapeau ← estZero(chiffres, n) MODULE: drapeau ← (chiffres[0] = 0) 2ième version:	*//version1: seulement les 4 premiers chiffres doivent être 0* `public static boolean estZero(int [ ] chiffres)` `{` `  boolean drapeau;` `  drapeau = chiffres[0] == 0;` `  return drapeau;` `}` *// version2: tous les 16 chiffres doivent être à 0* `public static boolean estZero(int [ ] chiffres)` `{` `  boolean drapeau;` `  drapeau =` `  return drapeau;` `}`
DONNÉES: (aucune) ITERMEDIATE: RÉSULTAT: tableauEntiers (référence à un tableau d'entiers) n (nombre d'éléments dans le tableau) EN-TÊTE: tableauEntiers, n ← readDigits( ) MODULE: AfficheLigne("Entrez le numéro de carte à l'aide de quatre ")	*/* Cette méthode demande à l'utilisateur d'entrer son numéro de carte de crédit à l'aide de 4 nombres, qui seront placés dans un tableau. Cette méthode fait appel à readIntLine( ) de la classe ITI1520 afin de lire le tableau d'entiers. */* `public static int [ ] litChiffres( )` `{` `  int [ ] tableauEntiers;` `  System.out.println` `  ("Entrez le numéro de carte à l'aide de quatre ");` `  System.out.println` `  ("nombres de quatre chiffres, séparés d'espaces;");` `  System.out.println` `  ("blancs, ou appuyez sur 0 pour terminer.");` `  tableauEntiers = ITI1520.readIntLine( );` `}`
DONNÉES: x (nombre entier) RÉSULTAT: somme (somme des chiffres du nombre entier) INTERMÉDIAIRES: (aucune) EN-TÊTE: somme ← sommeChiffres(x) MODULE:	*// Retourne la somme des chiffres d'un nombre x* `public static int sommeChiffres(int x)` `{` `  int somme = 0;` `  while (x != 0)` `  {` `  }` `  return somme;` `}`

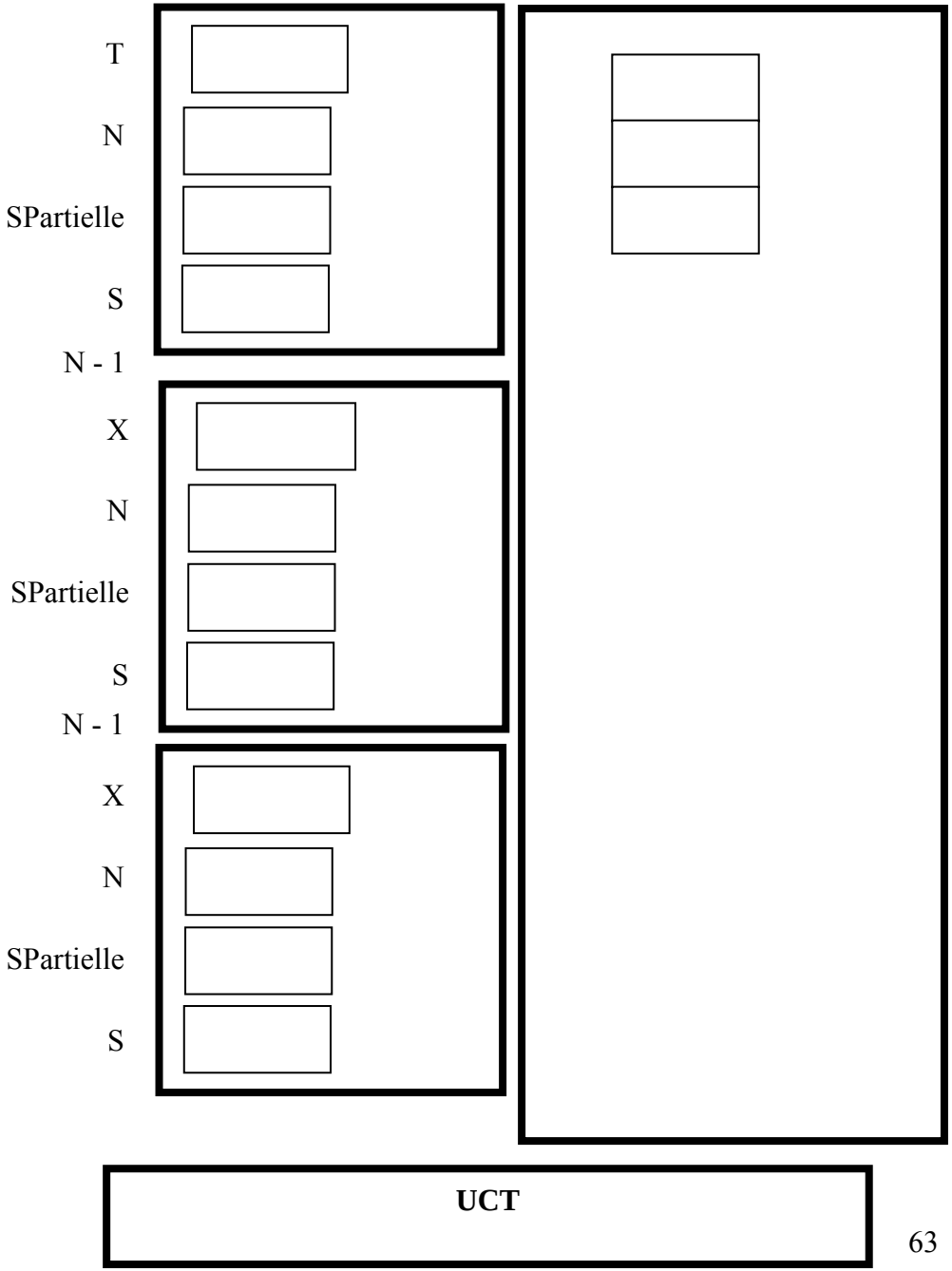
12. Section 8 - Exercices

Mémoire de programme **Exercice 8-1 – Algorithme pour somme récursive** **Mémoire de travail** **Mémoire globale**

Trace call SommeRéc(X, 3)

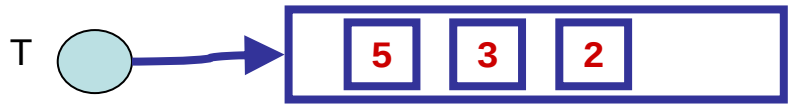
DONNÉES: T (référence à un tableau d'entiers)
N (nombre d'éléments à additionner dans le tableau)
RÉSULTAT: S (somme des N éléments du tableau)
INTERMÉDIAIRES:
SPartielle (somme partielle des N-1 éléments)
EN-TÊTE:
S ← SommeRéc(X, N)





Exercice 8-1 – Trace, Table 1 – SommeRéc(T, 3)

Instructions	Réf. par T	N	S	Somme
Valeurs initiales				



S ← SommeRéc(T,N-1)

Somme ← SommeRéc(T,N)

Exercice 8-1 – Trace, Table 2 – SommeRéc(T, 2)

Instructions	Réf. par T	N	S	Somme
Valeurs initiales				

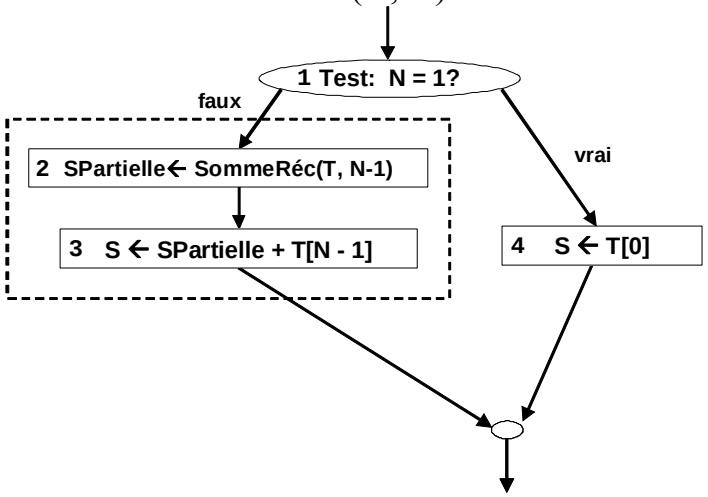
S ← SommeRéc(T,N-1)

Somme ← SommeRéc(T,N)

Exercice 8-1 – Trace, Table 3 – SommeRéc(T, 1)

Instructions	Réf. par T	N	S	Somme
Valeurs initiales				

DONNÉES: T (référence à un tableau d'entiers)
N (nombres d'éléments à additionner dans le tableau)
RÉSULTAT: S (somme des N elements)
INTERMÉDIAIRES:
SPartielle (somme de N-1 éléments dans le tableau)
EN-TÊTE:
S ← SommeRéc(X, N)




```
public static int sommeRec(int [] t,
                           int n)
{




}
```

UCT

Modèle algorithmique	Java
Exercice 8-3 (a) – Trouver X^N où X et N sont des entiers et $N \geq 0$, $X \geq 1$. (Version 1)	
DONNÉES: X (base) N (entier, puissance) RÉSULTAT: XàlaN (X à la puissance N) INTERMÉDIAIRES: M (vaut N-1; plus petit!) XàlaM (résultat partiel) EN-TÊTE: XàlaN ← Puissance (X,N) MODULE:	`// Méthode puissance: trouver X à la puissance N` `public static int puissance(int x, int n)` `{` `  // DÉCLARATION DE VARIABLES` `  int m;  // INTERMÉDIAIRE: valeur réduite` `  int xÀlaM;  // INTERMÉDIAIRE: résultat partiel` `  int xÀlaN;  // RÉSULTAT: résultat recherché` `  // MODULE DE L'ALGORITHME` `  // RETOURNE RÉSULTAT` `  return xÀlaN;` `}`

Modèle algorithmique	Java
Exercice 8-3 (b) – Trouver X^N où X et N sont des entiers et $N \geq 0$, $X \geq 1$. (Version 2 – efficace)	
DONNÉES: X (base) N (entier, puissance) RÉSULTAT: XàlaN (X à la puissance de N) INTERMÉDIAIRES: M (vaut N-1; plus petit!) XàlaM (résultat partiel) EN-TÊTE: XàlaN ← Puissance (X,N) MODULE:	`// MÉTHODE puissance: trouver X à la puissance N` `public static int puissance(int x, int n)` `{` `  // DÉCLARATION DE VARIABLES` `  int m;  // INTERMÉDIAIRE: valeur réduite` `  int xÀlaM;  // INTERMÉDIAIRE: résultat partiel` `  int xÀlaN;  // RÉSULTAT: résultat recherché` `  // MODULE D'ALGORITHME` `  // RETOURNE RÉSULTAT` `  return xÀlaN;` `}`

Modèle algorithmique	Java
Exercice 8-4 – Dans un tableau T de plus de N nombres, retourner VRAI si tous les nombres aux positions 0…N-1 de T sont égaux, sinon FAUX.	
DONNÉES: T (Référence à un tableau d'entiers) N (Nombre d'éléments à vérifier) RÉSULTAT: tousÉgaux (Booléen: VRAI si toutes les premiers N éléments ont la même valeur) INTERMÉDIAIRE: resPartiel (résultat partiel) EN-TÊTE: tousÉgaux ← égale(A, N) MODULE:	*// MÉTHODE égal: Détermine si les N premiers éléments de T* *// sont égaux.* `public static boolean égal (int [] t, int n)` `{` `  // DÉCLARATION DES VARIABLES / DICTIONNAIRE DE DONNÉES` `  boolean resPartiel; // INTERMÉDIAIRE: Résultat partiel` `  boolean tousÉgaux;  // RÉSULTAT: Résultat recherché` `  // MODULE DE L'ALGORITHME` `  // RÉSULTAT RETOURNÉ` `  return tousÉgaux;` `}`
Version efficace	*// MÉTHODE Égal: Détermine si les N premiers éléments de T* *// sont égaux.* `public static boolean égal (int [] t, int n)` `{` `  // DÉCLARATION DES VARIABLES / DICTIONNAIRE DE DONNÉES` `  boolean tousÉgaux;  // RÉSULTAT: Résultat recherché` `  // MODULE DE L'ALGORITHME` `  return tousÉgaux;  // RÉSULTAT RETOURNÉ` `}`

Modèle algorithmique	Java
Exercice 8-5 – Calcule N !.	
DONNÉES: N *(entier)* RÉSULTAT: F *(entier, N factoriel)* INTERMÉDIARE: (aucune) EN-TÊTE: F ← Fact(N) MODULE:	*// Method fact* *// Given: n, an integer* public static int fact(int n) { int f; *// RÉSULTAT* return f; }
Exercice 8-6 – Trouver la somme de 1+2+…+N.	
DONNÉES: N (un entier) RÉSULTAT: sommeN (somme des entiers de 1 a N) INTERMÉDIAIRE: M (vaur N-1; plus petit!) sommeM (somme des entiers de 1 à M) EN-TÊTE: sommeN ← sommeJusquàN(N) MODULE:	public static int sommeJusquàN(int n) { // Déclarations de variables int sommeN; // RÉSULTAT : la somme de 1 à N int m; // INTERMÉDIAIRE : vaut n-1 int sommeM; // INTERMÉDIAIRE : la somme de 1 à M // Retourne résultat return(sommeN); }

Modèle algorithmique	Java
Exercice 8-7 – Inverser l'ordre des caractères dans un tableau A de N caractères.	
DONNÉES: t (référence à un tableau de caractère à inverser) bas (index bas) haut (index haut) RÉSULTATS: (aucun – tableau référé est modifié) MODIFIED: A (réfère au tableau inversé) INTERMÉDIAIRES: nouvHaut (nouveau index haut) nouvBas (nouveau index bas) temp (temporaire pour faire l'échange) EN-TÊTE inverse(t, bas, haut) MODULE:	`// MÉTHODE Échange: Inverser les caractères dans un tableau` `// référé par t de taille N.` `// À invoquer initialement avec Échange(t, 0, N-1).` `public static void inverse (char [] t, int bas, int haut)` `{` `   // DÉCLARATION DES VARIABLES / DICTIONNAIRE DE DONNÉES` `   int nouvHaut;  // INTERMÉDIAIRE: Haut plus petit` `   int nouvBas;   // INTERMÉDIAIRE: Bas 'plus petit'` `                  //                    (plus grand!)` `   char temp;     // INTERMÉDIAIRE: Caractère tampon` `   // MODULE DE L'ALGORITHME` `   // RÉSULTAT RETOURNÉ: Le tableau 't' est déjà modifié!` `}`

Modèle algorithmique	Java
Exercice 8-8 – Trier un tableau A de N nombres en ordre croissant.	
DONNÉES: T (Référence au tableau d'entiers à trier) N (entier, 0 à N-1 pour indexé T) RÉSULTATS: (aucun) MODIFIED: T (référence au tableau trié) INTERMÉDIAIRES: PosMax (position de la plus grande valeur dans T[0] à T[N-1]) Temp (valeur temporaire pour échange) EN-TÊTE Trier(T, N) MODULE:	`public static void trier(int[] t, int n)` `{` `  // DÉCLARATION DE VARIABLES` `  // DONNÉES: t – référence au tableau à trier` `  //  n – nombres d'éléments à trier - notez` `  //  que t.length NE peut PAS être utilisé.` `  // INTERMÉDIAIRES` `  int posMax;  // position de la plus grande valeur` `  int temp; // utliser pour échange` `  // MODULE` `}`

Modèle algorithmique	Java
Exercice 8-8 – Trier un tableau *A* de N nombres en ordre croissant: – algorithme/méthode **trouveValeurMaximale**	
DONNÉES: T (référence à un tableau d'entiers) N (entier, 0 à N-1 pour indexé T) RÉSULTATS: Pos (position de l'élément maximal de T[0] à T[N-1]) INTERMÉDIAIRES: M (entier, intervalle plus petit!) PosM (position de l'élément maximal dans T[0] à T[M-1) EN-TÊTE Pos ← TrouveValeurMaximale(T, N) MODULE :	`public static void trouveValeurMaximale(int[] t, int n)` `{` `  // DÉCLARATION DE VARIABLES` `  // DONNÉES: t – référence au tableau` `  //  n – nombres d'éléments à chercher - notez` `  //  que t.length NE peut PAS être utilisé.` `  int pos; // RÉSULTAT-position de la plus grande valeur` `  // INTERMÉDIAIRES` `  int m;  // plus petit` `  int posM; // utiliser pour échange` `  // MODULE` `  // RÉSULTAT` `  return(pos);` `}`

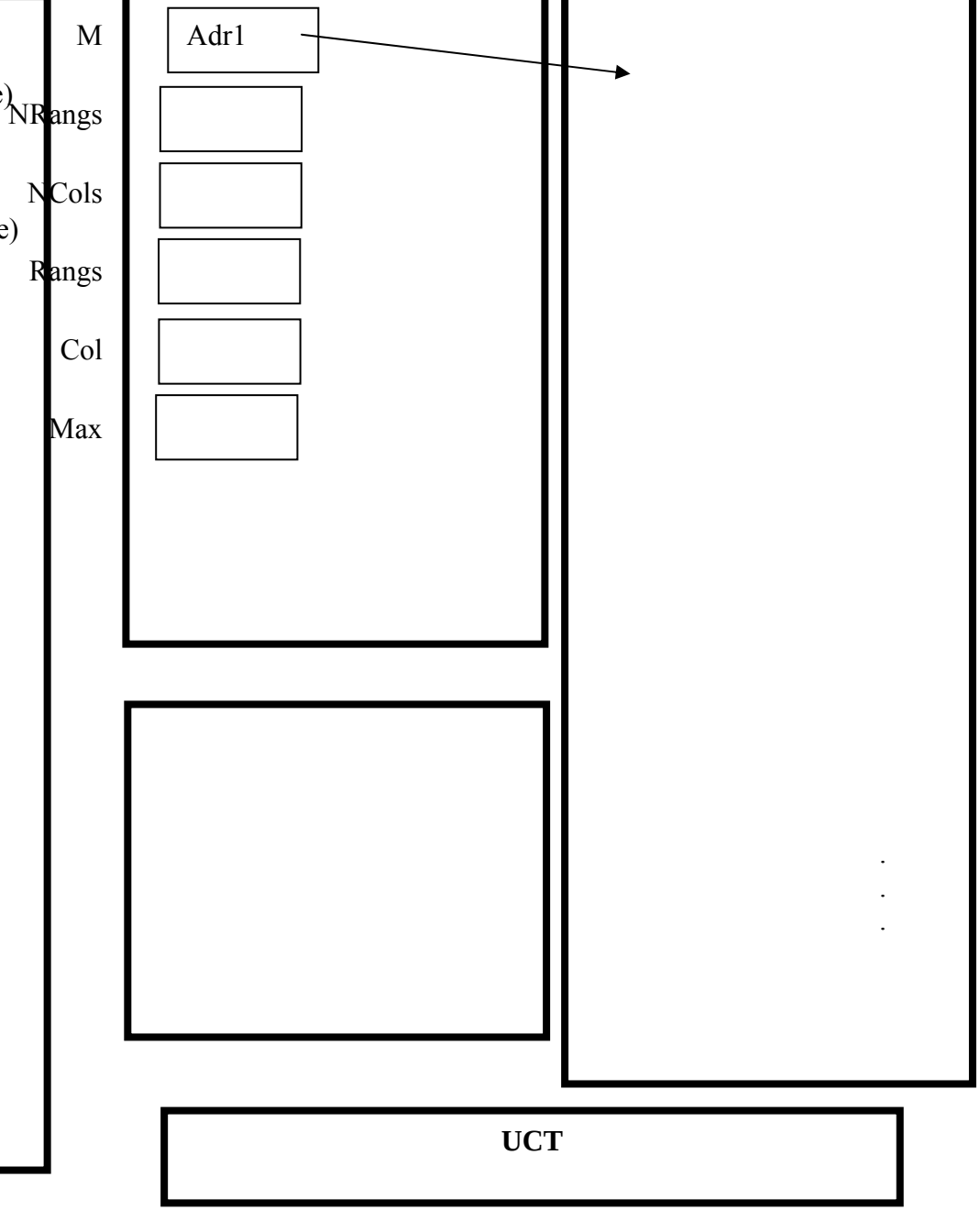
13. Section 9 Exercices

- Exercice 9-1: La matrice M est un tableau de 4 tableaux, chacun ayant 6 éléments. Ainsi:
  M[1][2] vaut
  M[2][5] vaut
  M[4][1] vaut
  M[3] vaut

Mémoire de programme **Exercice 9-2 – Valeur maximale dans une matrice** **Mémoire de travail** **Mémoire globale**

DONNÉES: M (référence à une matrice)
NRangs (nombres de rangées dans matrice)
NCols (nombres de colonnes dans matrice)
INTERMÉDIAIRES:
Rang (index de rangée courante)
Col (index de colonne courante)
RÉSULTAT: Max (la valeur maximale dans la matrice)
EN-TÊTE: Max ← TrouveMaxMatrice(M, NRangs, NCols)
MODULE:

Alternative:



Mémoire de programme **Exercice 9-3 – Algorithme VérifDiag** **Mém de travail** **Mém globale**

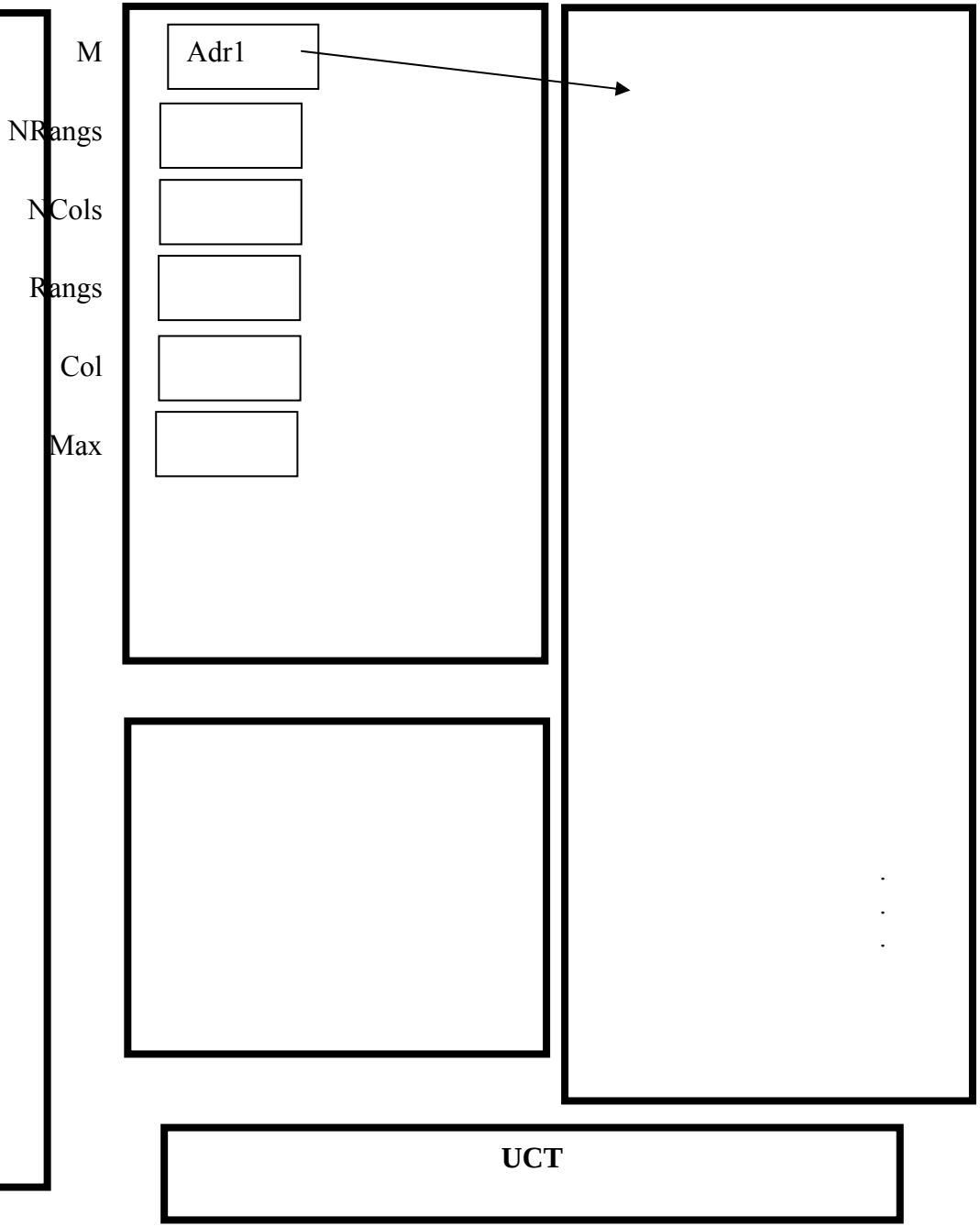
DONNÉES: M (référence à une matrice)
NRangs (nombre de rangées dans matrice)
NCols (nombre de colonnes dans matrice)
INTERMÉDIAIRES:
Rangs (index de la rangée courante)
Col (index la colonne courante)
RÉSULTAT: estDiagonale (Booléen: vrai si matrice est diagonale)
EN-TÊTE: estDiagonale ← VérifDiag(M, NRangs, NCols)
MODULE:

(Version efficace)

UCT

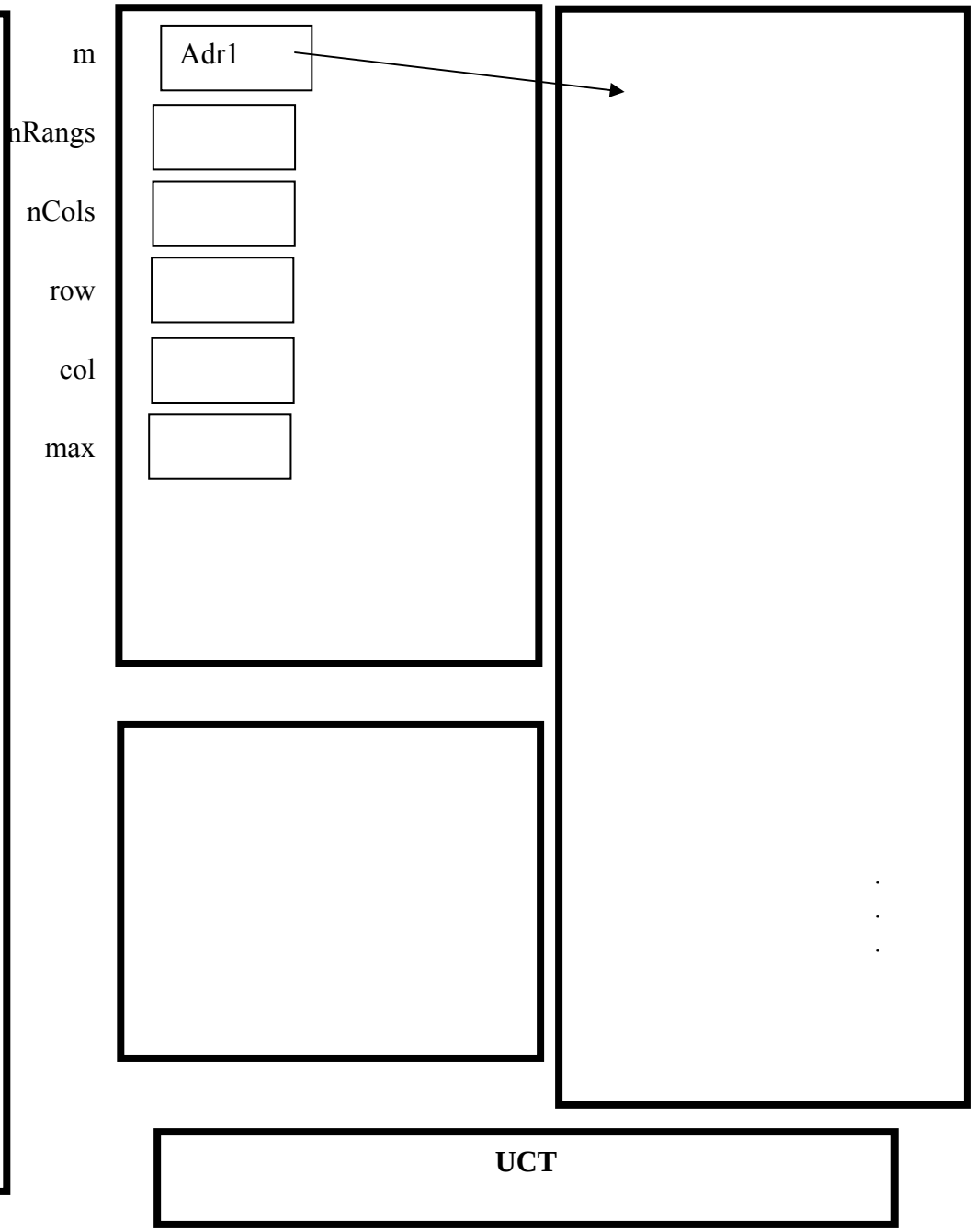
Mémoire de programme **Exercice 9-4 – Valeur maximale en Java** **Mémoire de travail** **Mémoire globale**

```
// Note:  Integer.MIN_VALUE est la valeur
// entière la plus négative permise pour un
// int Java, et peut être utilisé pour –∞.

public static int maxMatrice (int[][] m,
                    int nRangs, int nCols)
{
   int max = Integer.MIN_VALUE;
     // RÉSULTAT. m[0][0] pourrait être une
    // autre option… si la matrice M n'est
   //  pas vide!
   int rang;   // INTERMÉDIAIRE
   int col;    // INTERMÉDIAIRE
   // Module









   return max;
}
```



Mémoire de programme **Exercice 9-5 – Lecture d'une matrice** **Mémoire de travail** **Mémoire globale**

```
public static int[][] litMatrice(
                    int nRangs, int nCols)
{
  int rang; // INT: index pour les rangées
  int col;  // INT: index pour les colonnes
  int[][] m ; // RÉSULTAT: Matrice lue.









  return m;
}
```



Mémoire de programme

Exercice 9-6 – Trouver le vol direct le moins cher

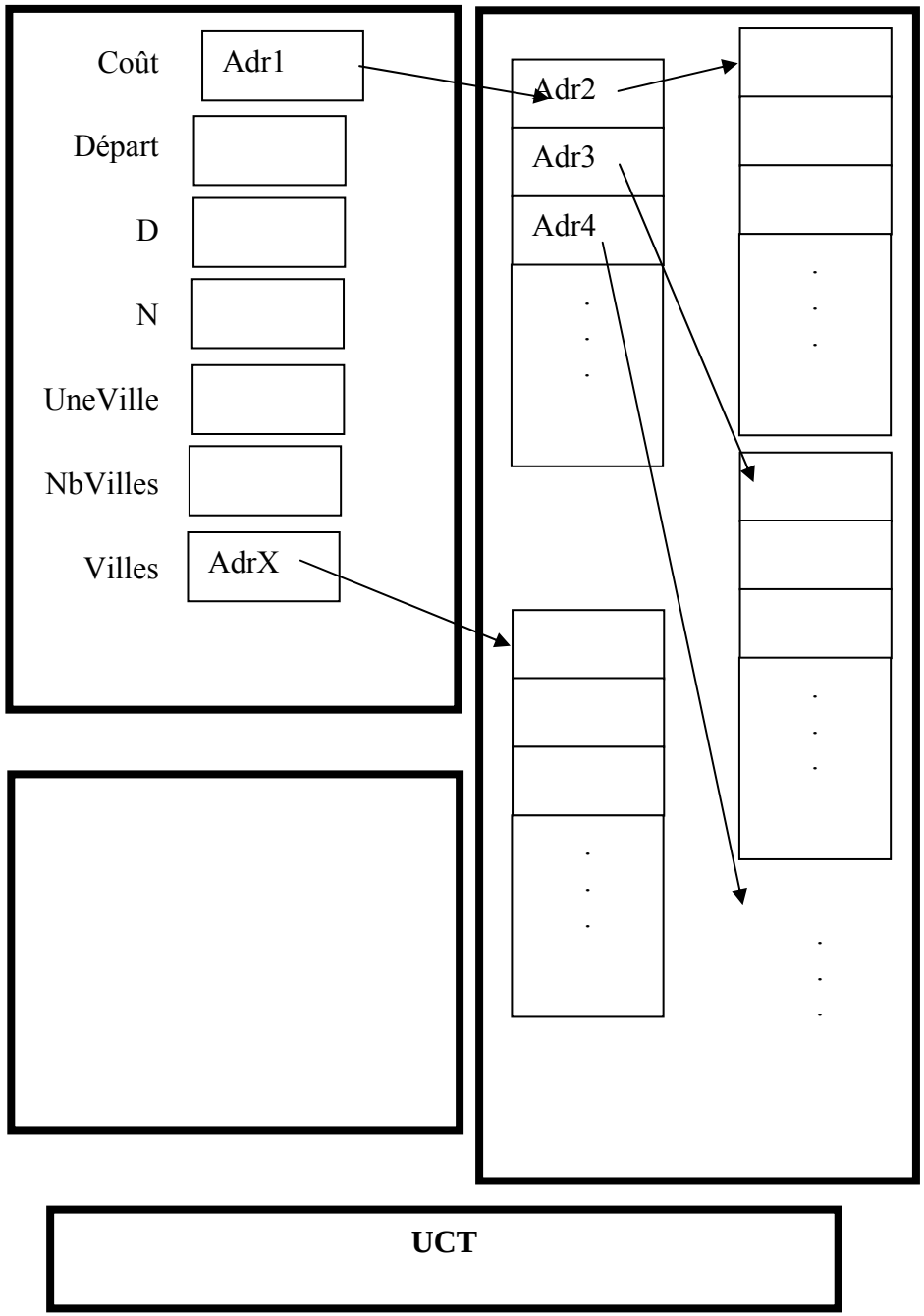
DONNÉES:
- Départ (le numéro de la ville où vous êtes)
- Coût (référence à la matrice de coûts)
- D (votre budget)
- N (le nombre total de villes)

RÉSULTATS:
- Villes (référence à un tableau de villes visitables)

INTERMEDIATE:
- UneVille (la destination présentement considérée)
- NbVilles (le nombre de villes visitables)

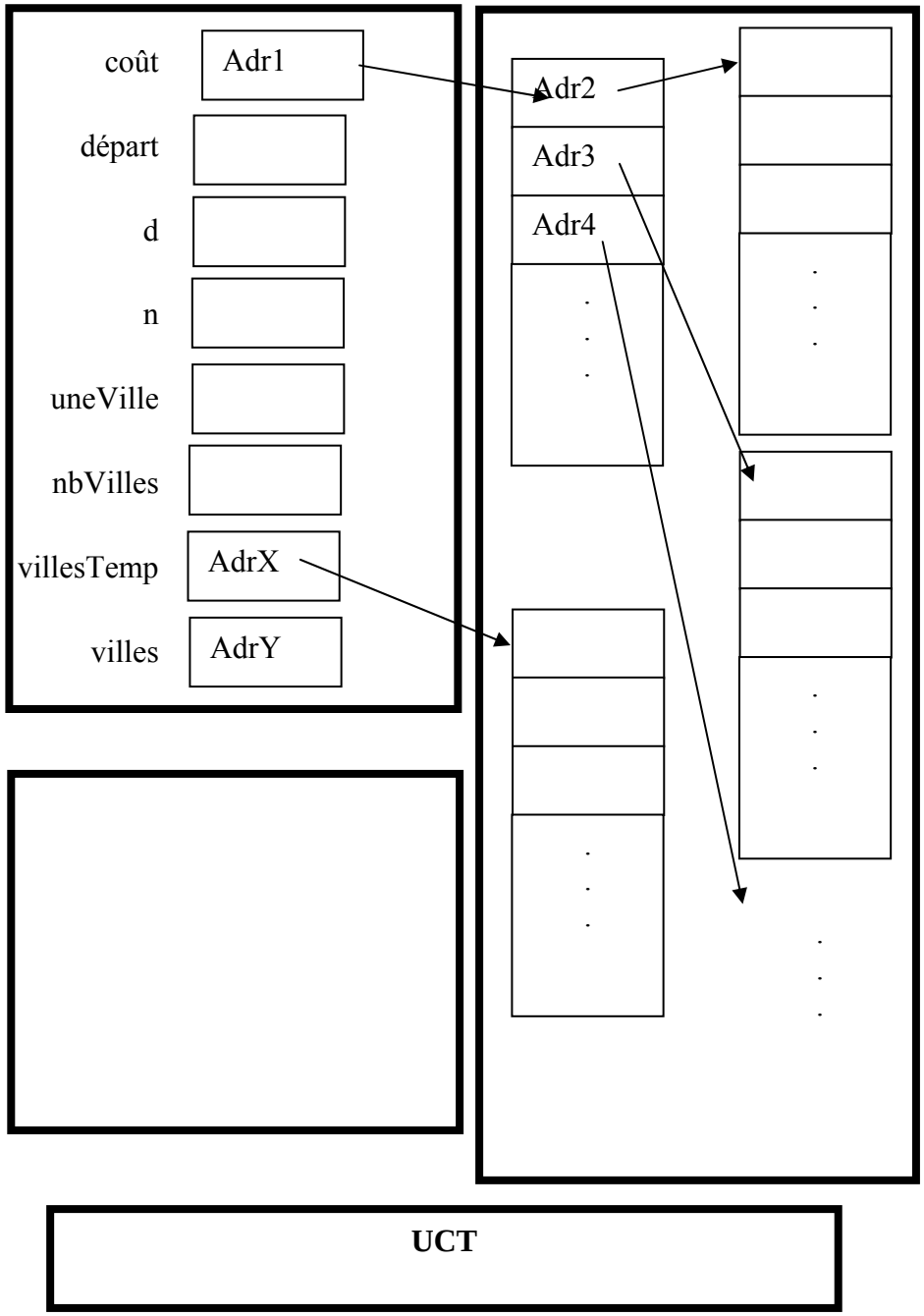
EN-TÊTE

(NbVilles, Villes) ← VolsPasChers (Départ, Coût, D)



Mémoire de programme **Exercice 9-6 (suite) – Traduction en Java** **Mémoire de travail** **Mémoire globale**

```
public static int[] volsPasCher ( int départ,
                  int[][] coût, int d, int n )
{
   int uneVille; //INT, index de la ville con.
   int nbVilles; //INT, # de villes visitables
   int[] villes; //RÉS, tableau de villes visit

   // MODULE DE L'ALGORITHME









   // RÉSULTAT: Maintenant, le résultat peut
   // être retourné…
   return villes;
}
```



Mémoire de programme **Exercice 9-7**– Algorithme EffaceRang **Mém de travail** **Mém globale**

DONNÉES**:** M *(référence à une matrice carrée)*
N *(nombre de rangées et de colonnes dans matrice)*
R *(numéro de la rangée à effacer)*
RÉSULTAT**:** Aucun
MODIFIÉE**:** M *(rangée R est effacée, les autres rangées sont déplacées, et la dernière rangées est mise à 0)*
INTERMÉDIAIRE**:** Index *(indice de la rangée à déplacer)*
EN-TÊTE : EffaceRang(M, N, R)

```
public static void effaceRang(int [][]m,
                              int n, int r)
{
  int index; // INTERMÉDIAIRE
```

DONNÉES**:** M *(une matrice carrée)*
N *(taille de M)*
R *(numéro de la rangée à déplacer)*
RÉSULTAT**: (**Aucun)
MODIFIÉE**:** M *(rangée R copiée à rangée R-1)*
INTERMÉDIAIRE**:** Index *(indice de la colonne)*
EN-TÊTE: DéplaceHaut(M, N, R)

```
}

private static void déplaceHaut(int [][] m,
                                int n, int r)
{
  int index; // INTERMÉDIAIRE
```

```
}
```

UCT

DONNÉES: M *(une matrice carrée)*
N *(taille de M)*
RÉSULTAT: Aucun
MODIFIÉE: M *(dernière rangée mise à 0)*
INTERMÉDIAIRE: Index *(index d'une colonne)*
EN-TÊTE: MetRangÀZéro(M, N)

```
private static void metRangÀZéro(int [ ][ ]m,
                                 int n)
{
  int index; // INTERMÉDIAIRE




}
```

UCT

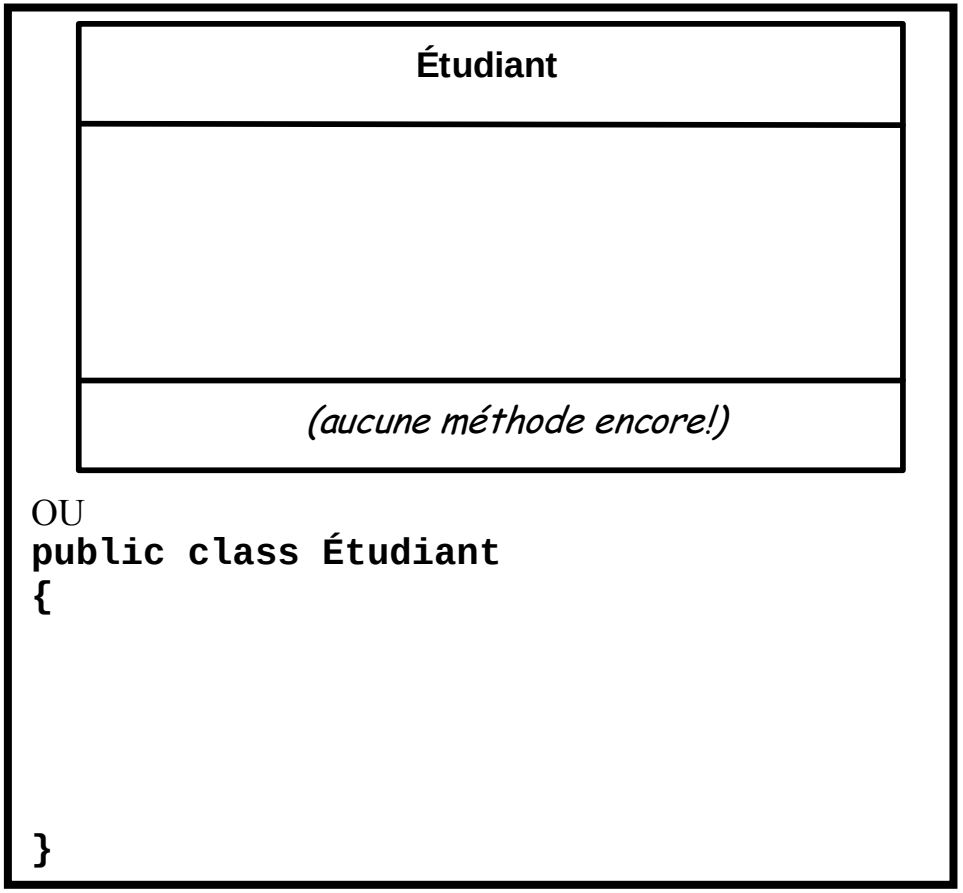
14. Section 10 - Exercices

- **Exercice 10-1:** Qu'est-ce qui ne va pas avec les solutions suivantes:
  - Chaque valeur est une variable séparée:

  - Mettre toutes les valeurs dans un tableau:

Mémoire de programme

Exercice 10-2 – Première version de la classe Étudiant

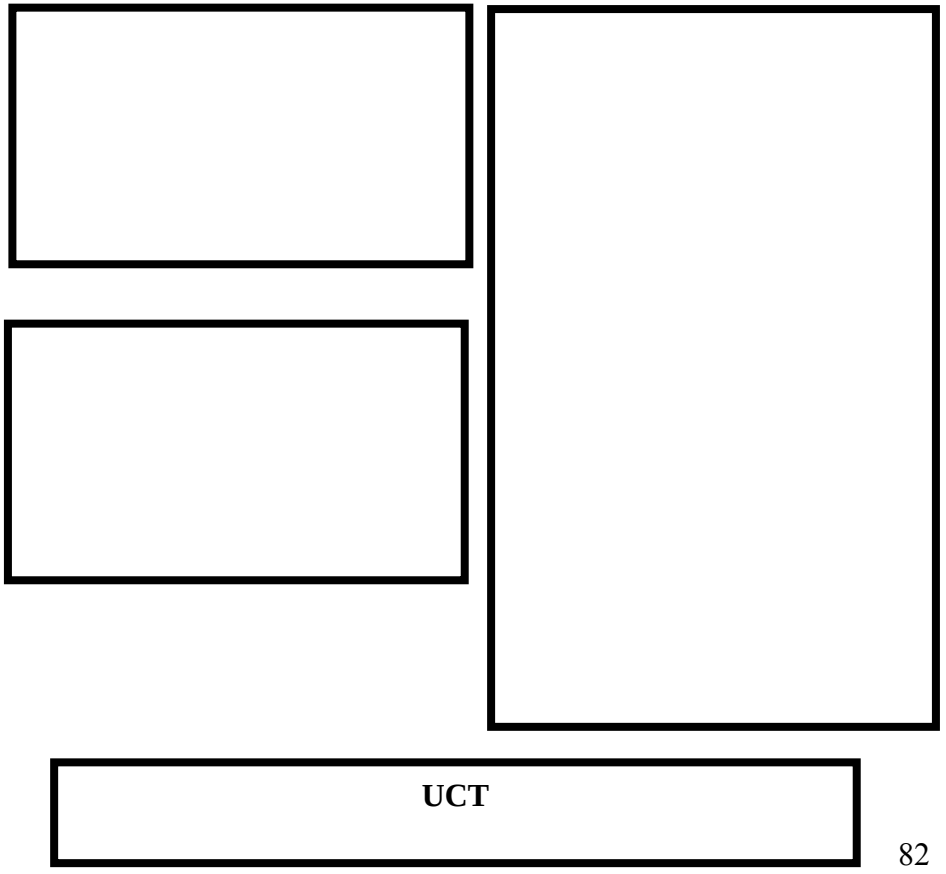


OU

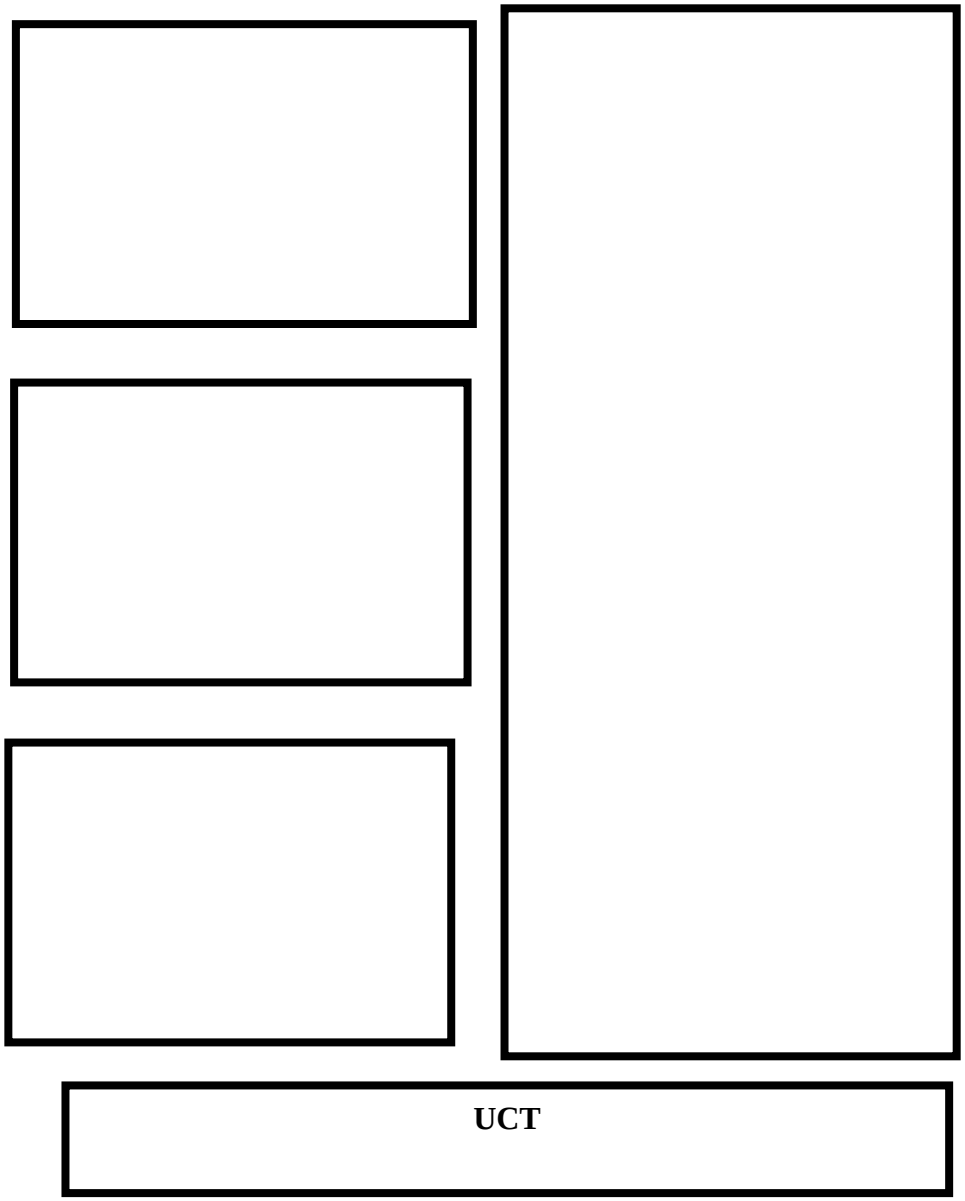
```
public class Étudiant
{




}
```

Mémoire de travail

Mémoire globale

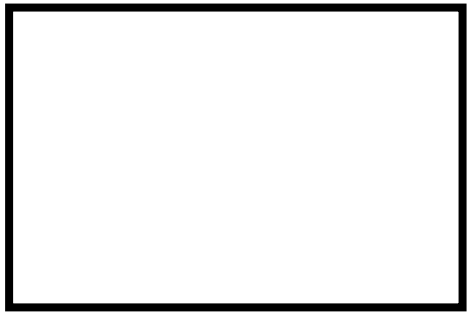


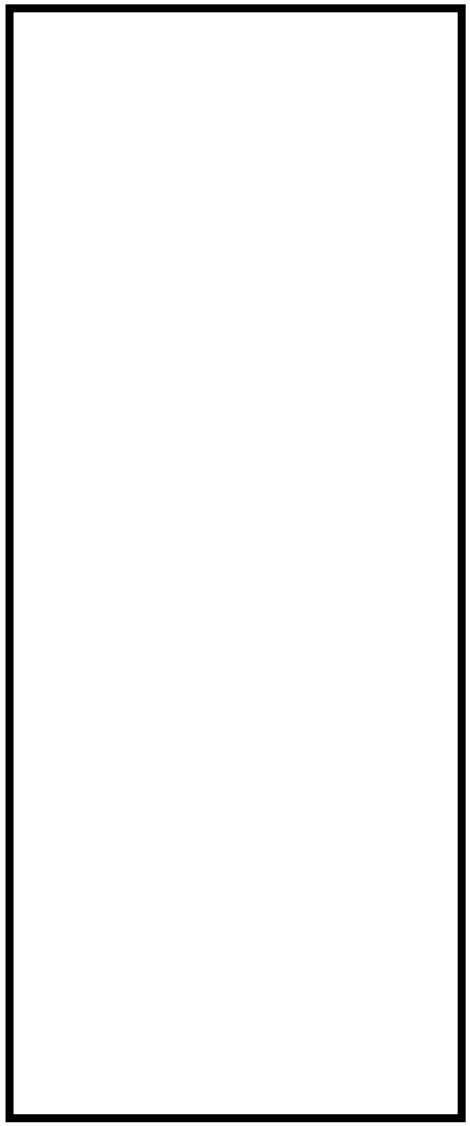
```
public class Section10
{
   public static void main(String [] args)
   {
      Étudiant unÉtudiant;
      unÉtudiant = new Étudiant();
      unÉtudiant.numÉtud = 1234567;
      unÉtudiant.examPartiel = 60.0;
      unÉtudiant.examFinal = 80.0;
      unÉtudiant.créditable = true;

      Étudiant moiAussi = new Étudiant();
      moiAussi.numÉtud = 1069665;
      moiAussi.examPartiel = 73.0;
      moiAussi.examFinal = 79.0;
      moiAussi.créditable = false;
   }

}
```



Mémoire de programme | Information Hiding in the Student Class | Mémoire de travail | Mémoire globale

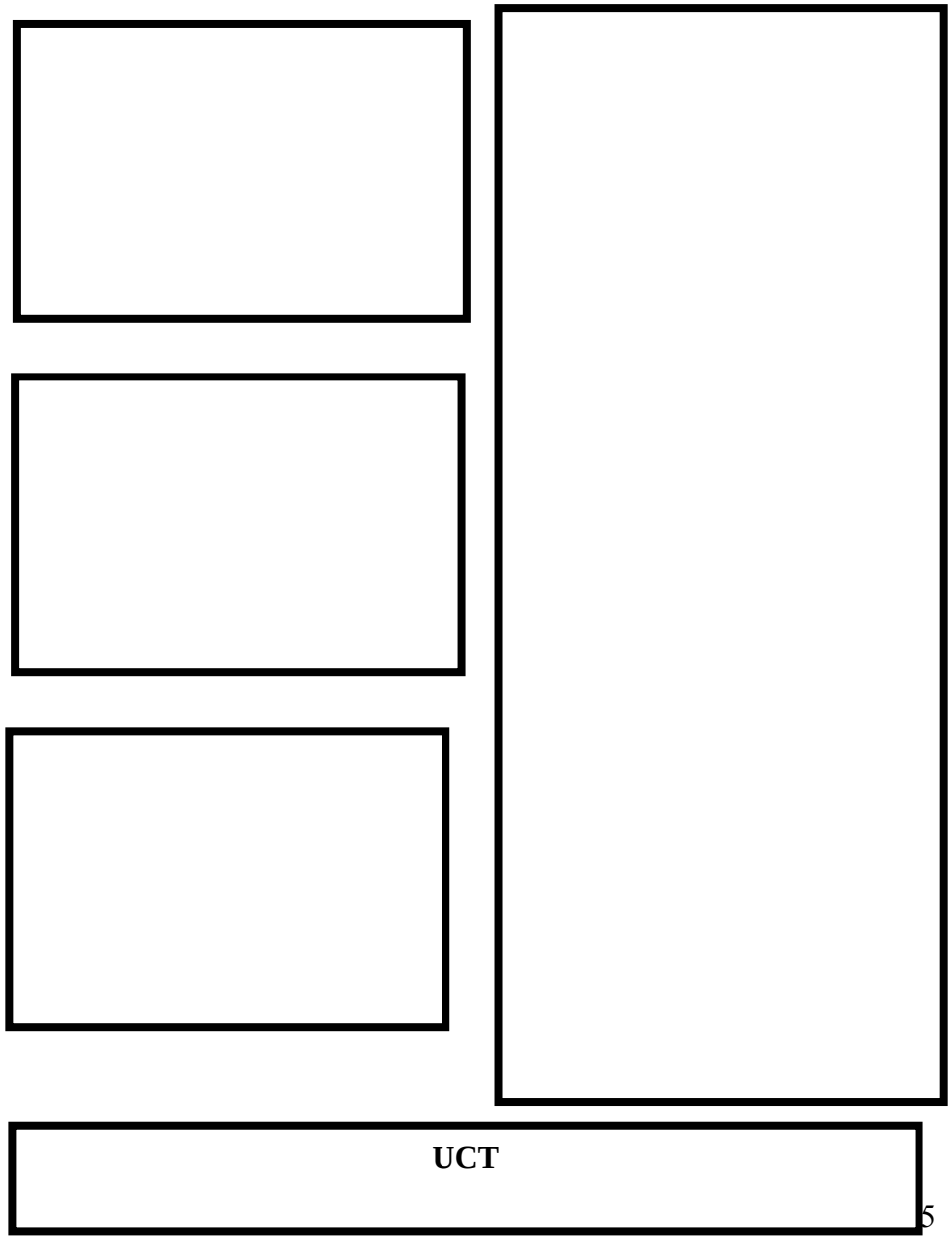
```
public class Étudiant
{
   // Attributs
   private int numÉtud;
   private double examPartiel;
   private double examFinal;
   private boolean créditable;
   private double noteFinale;

   // Méthodes
   public int getNumÉtud()
   { // insérez votre code ici }
   public void setNumÉtud( int nouvNum )
   { // insérez votre code ici }
   public double getExamPartiel()
   { // insérez votre code ici }
   public void setExamPartiel(double nouvNote)
   { // insérez votre code ici }
   public double getExamFinal()
   { // insérez votre code ici }
   public void setExamFinal( double nouvNote )
   { // insérez votre code ici }
   public boolean getCréditable()
   { // insérez votre code ici }
   public void setCréditable(boolean nouvValeur)
   { // insérez votre code ici }
   public double getNoteFinale()
   { // insérez votre code ici }
   private void recalculeNoteFinale()
   { // insérez votre code ici }
} // fin de la classe Étudiant
```





Mémoire de programme **Méthodes dans la classe Étudiant** **Mémoire de travail** **Mémoire globale**

```
public class Étudiant
{
   // attributs et autres méthodes...

   public void setExamPartiel(double nouvNote)
   {
      this.examPartiel = nouvNote ;
      this.recalculeNoteFinale( );
   }

   public void setExamFinal(double nouvNote)
   {
      this.examFinal = nouvNote ;
      this.recalculeNoteFinale();
   }

   private void recalculeNoteFinale ()
   {
      this.noteFinale = 0.2 * this.examPartiel
                        + 0.8 * this.examFinal;
   }
}
```

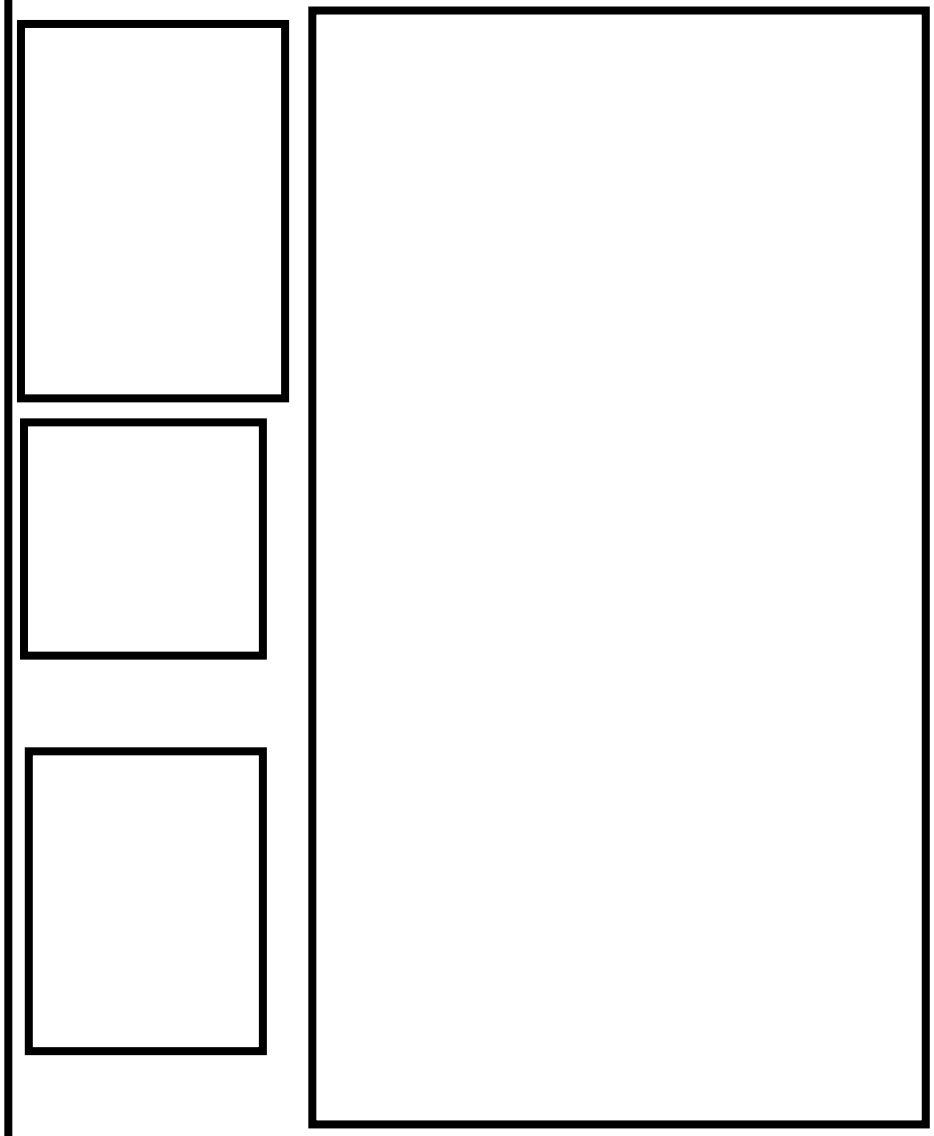


UCT

15. Section 11 - Exercices

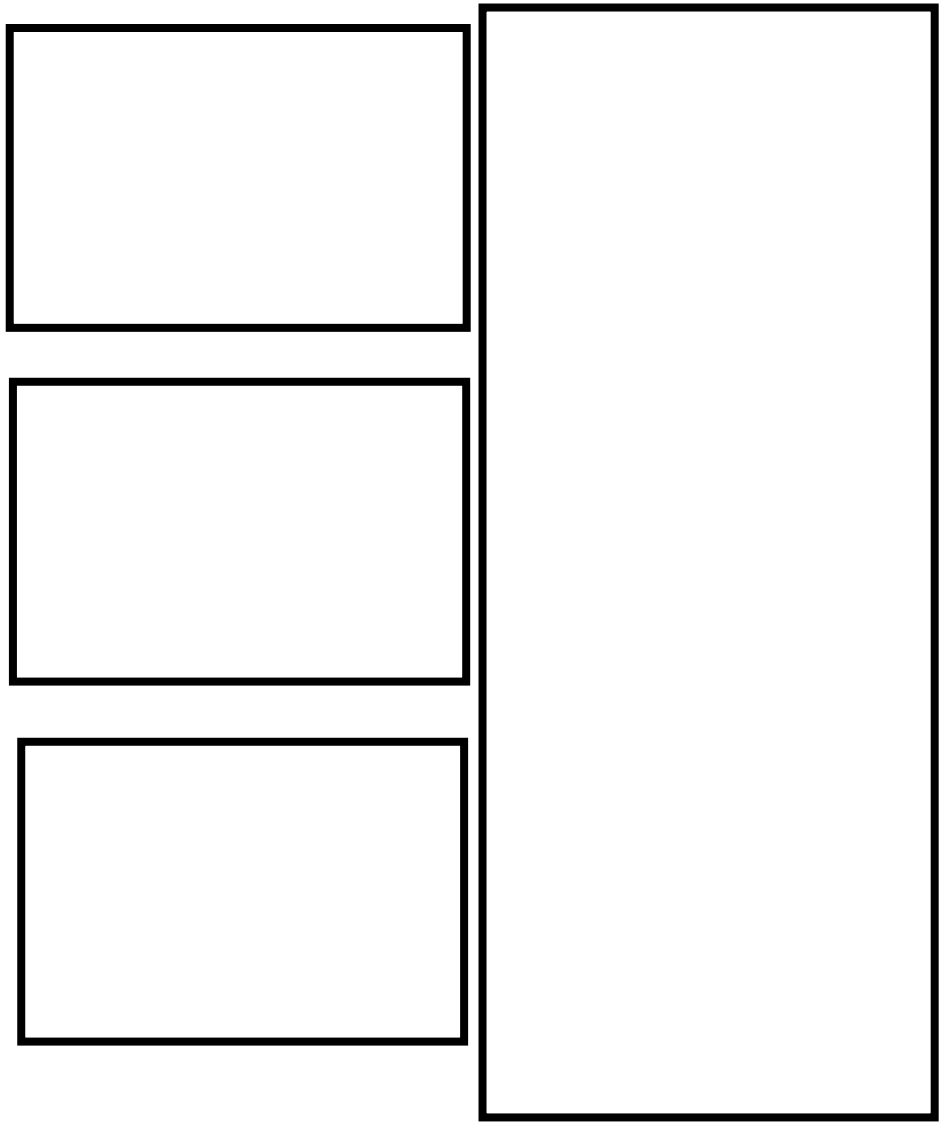
Mémoire de programme **Exercice 11-1 – Utilisation des Constructeurs** **Mém de travail** **Mémoire globale**

```
public class Section11
{
   public static void main(String [] args)
   {
     Étudiant étudiantA; // variable de référence
     Étudiant moiAussi;  // autre variable de réf.
     Étudiant étudiantB; // une 3ième variable de réf
               •
               •
     étudiantA = new Étudiant(1234567,60.0,80.0,true);
     moiAussi = new Étudiant(7654321,true);
     étudiantB = étudiantA;
               •
               •
               •
   }
}

class Étudiant
{
   // ... Les champs seraient définis ici ...
   public Étudiant(int n, double ePartiel,
                   double eFinal, boolean crédit)
   {
      numÉtud = n;
      examPartiel = ePartiel;
      examFinal = eFinal;
      créditable = crédit;
   }
   public Étudiant(int n, boolean crédit )
   {
      numÉtud = n;
      examPartiel = 0.0;    // une valeur « sûre »
      examFinal = 0.0;      // une valeur « sûre »
      créditable = crédit;
   }
   // ... Autres méthodes ...
```



Mémoire de programme **Exercice 11-2 – Calcule de la note finale** Mémoire de travail Mémoire globale

```
public class Student
{
   // Attributes
   public int numÉtud;
   public double examPartiel;
   public double examFinal;
   public boolean créditable;
   private double [] devoirs;
   // Methods
   private double calcMoyDevoirs()
   {









   }

   public double getNoteFinale()
   {









   }
} // fin de la classe Étudiant
```



UCT

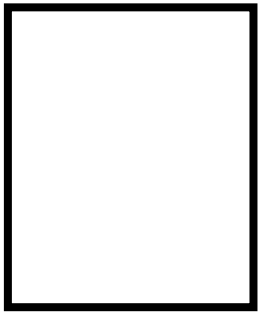
Mémoire de programme **Exercice 11-3 – Tableau d'objets** Mémoire de travail Mémoire globale

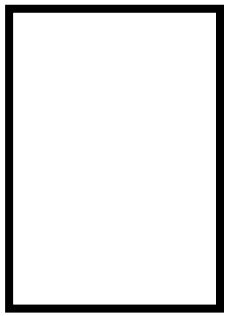
```
public class Section11
{
   public static void main(String [] args)
   {
     Cours unCours;
     unCours = new Cours("ITI1520",
                         "Intro. à l'info.");
     unCours.ajoutÉtudiant(123456,true);
     unCours.ajoutÉtudiant(654321,false);
   }
}
```

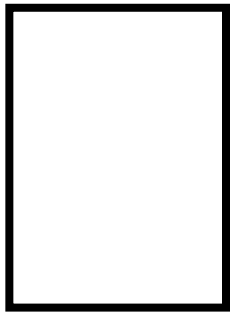
Cours

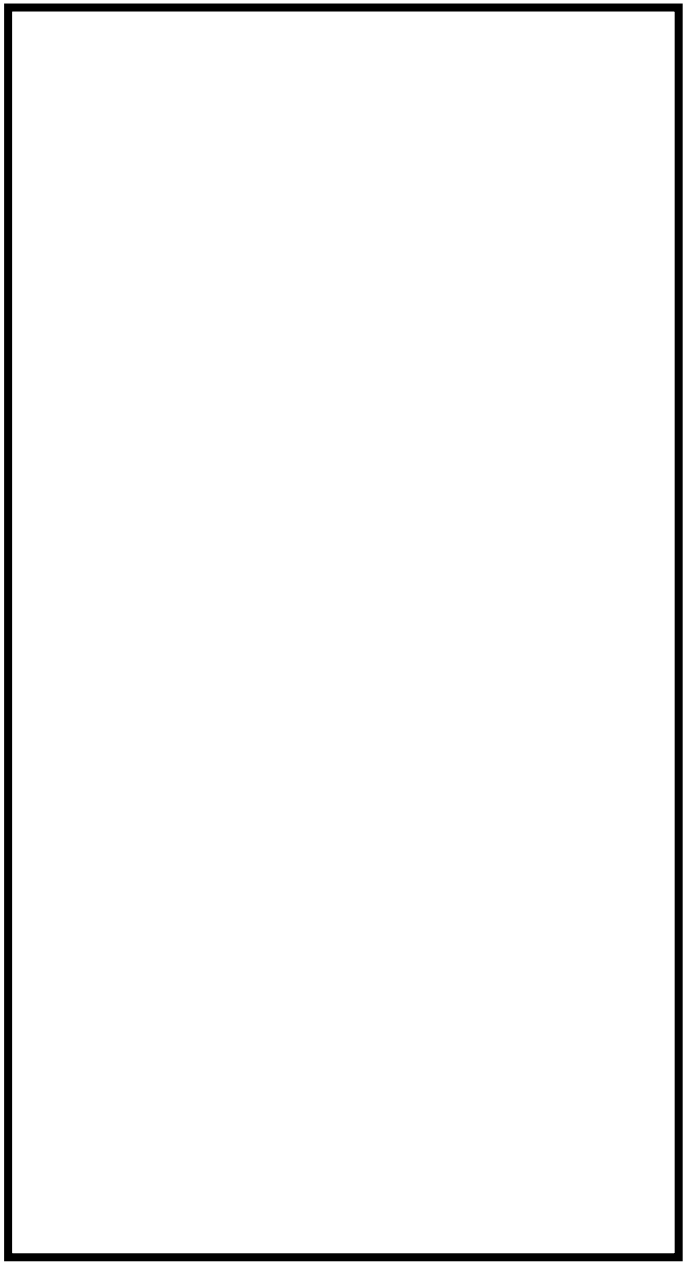
– cote : String
– titre : String
– étudiants: Etudiant [ ]
…

+ Cours(cote : String, titre : String)
+ ajoutÉtudiant(numÉtud : int, créditable : boolean)







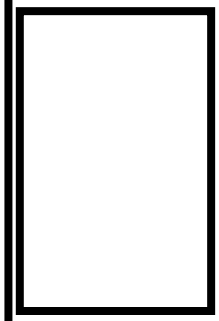


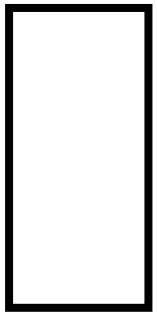
UCT

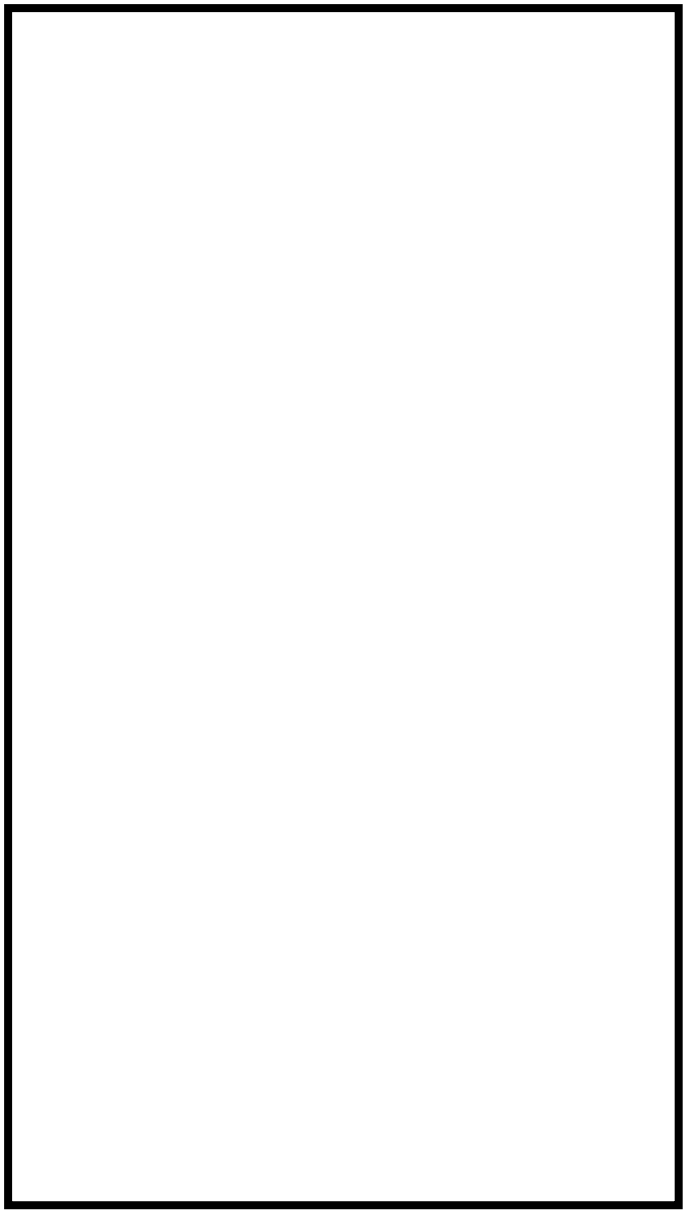
Mémoire de programme **Exercice 11-4 – Utilisation des variables et méthodes de classe** **Mémoire globale**

```
public class Section11
{
   public static void main(String [] args)
   {
     int numDev;
     Étudiant étudiantA; // variable de référence
     Étudiant moiAussi;  // autre variable de réf.
     étudiantA = new Étudiant(1234567,60.0,79.0,true);
     moiAussi = new Étudiant(7654321,54.5,83.4,true);
     for(numDev=0 ; numDev<5 , numDev=i+1)
     {
        étudiantA.setDevoirs(numDev, 60.0);
        moiAussi.setDevoirs(numDev, 65.0);
     }
     System.out.println("La note pour étudiant "
                     +étudiantA.getNumÉtud()+" est "
                     + étudiantA.getNoteFinale();
     Étudiant.setPoidsPartiel(0.30);
     Étudiant.setPoidsDevoirs(0.15);
     System.out.println("La note pour étudiant "
                     +moiAussi.getNumÉtud()+" est "+
                      moiAussi.getNoteFinale();
     System.out.println("La note pour étudiant "
                     +étudiantA.getNumÉtud()+" est "
                     + étudiantA.getNoteFinale();
   }
}
```

Mémoire de travail







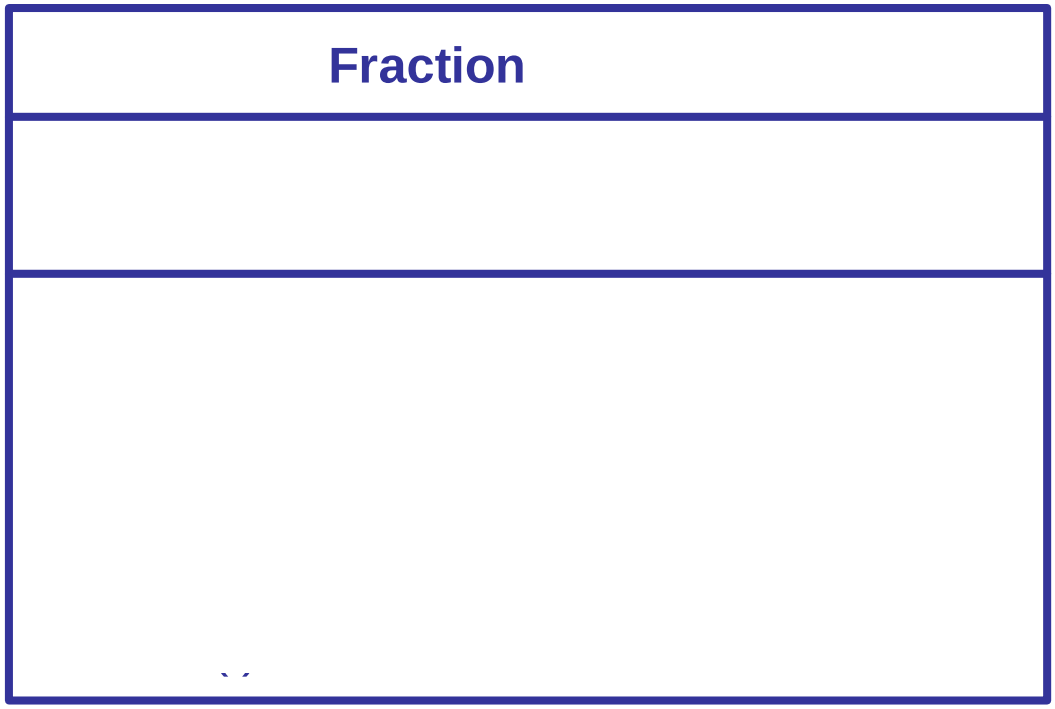
Fenêtre de console

UCT

Exercice 11-5 – Conception de la classe Fraction

- Quelle information avons-nous besoin d'emmagasiner dans une Fraction?

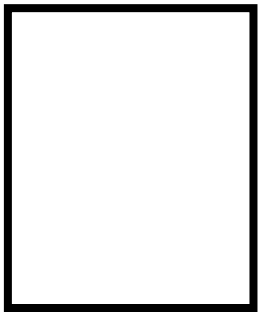
- De quelles opérations avons-nous besoin?
  - Nous nous limiterons à l'addition comme opération mathématique

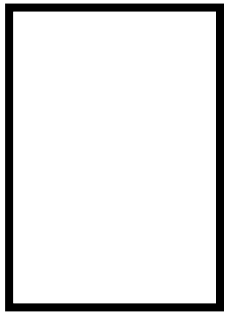


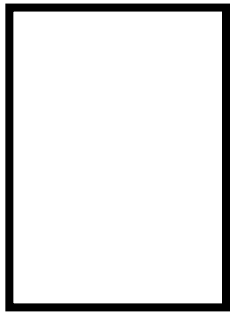
Mémoire de programme **Exercice 11-6 – Réduction d'une Fraction** **Mémoire globale**

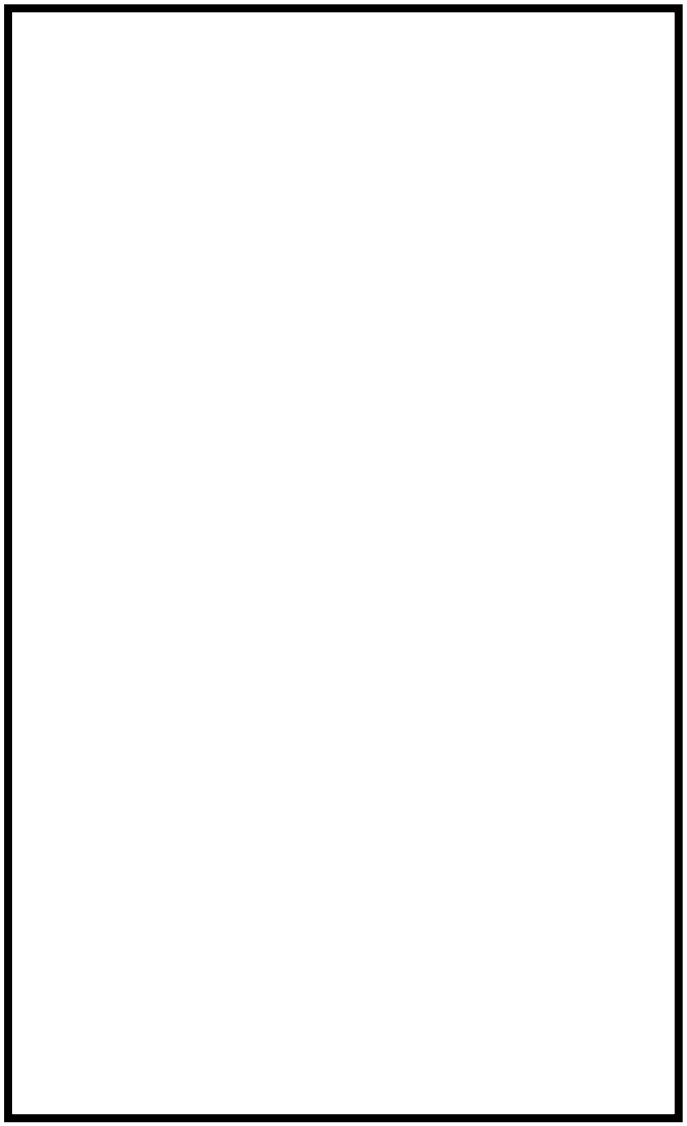
```
// méthode d'instance
private void réduit( )
{




}
```

Mémoire de travail







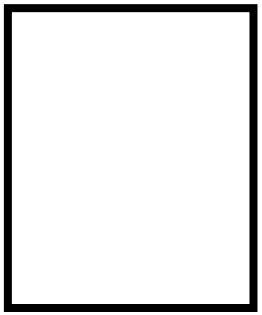


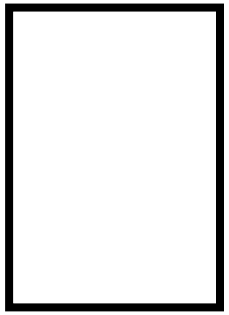
UCT

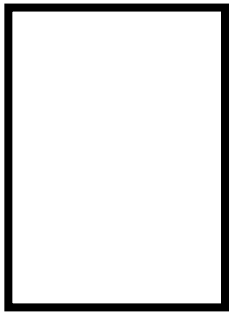
Mémoire de programme **Exercice 11-7 – Méthode pour PGCD** **Mémoire globale**

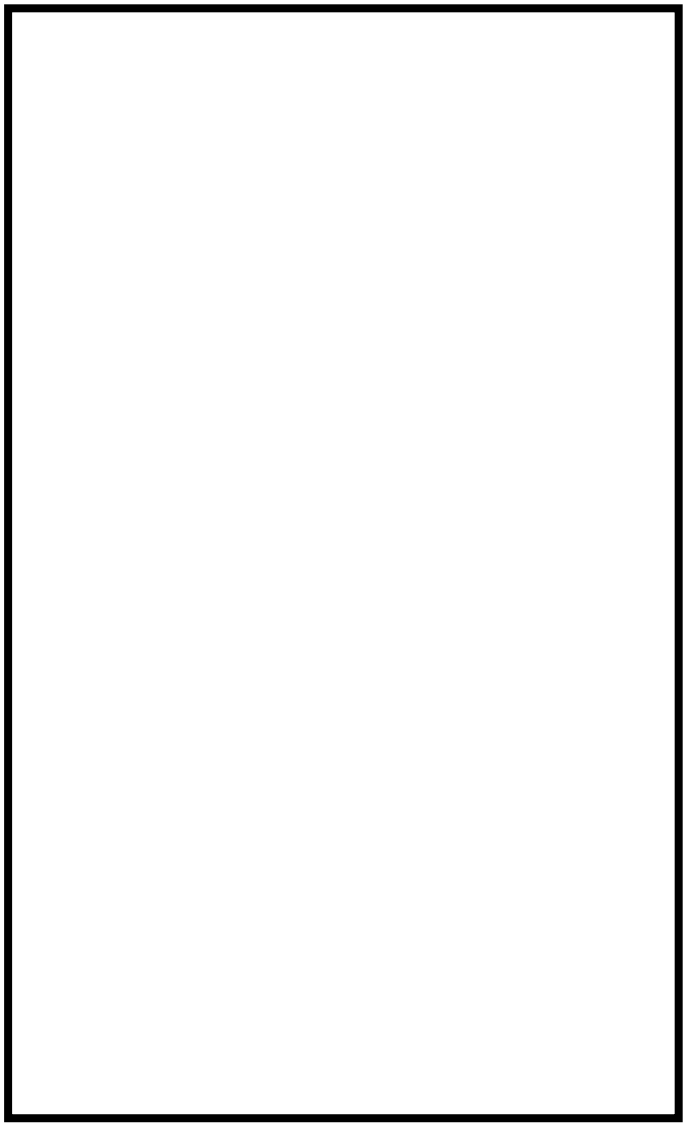
```
// méthode de classe
private static int pgcd (int a, int b)
{




} // notez que pgcd(0, 0) == 0
```

Mémoire de travail









Mémoire de programme

Mémoire globale

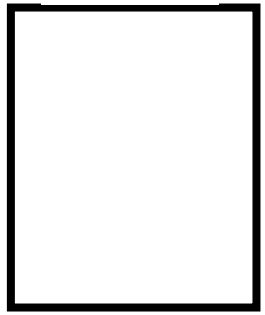
Exercice 11-8 – Constructeurs de Fraction

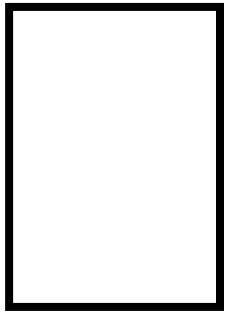
```
public Fraction(int n, int d)
{




}

public Fraction(int entier)
{


}
```

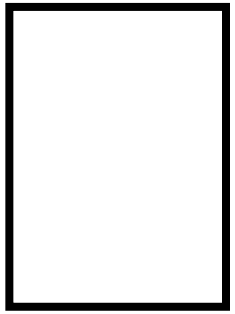
Exercice 11-9 Affichage des Fractions

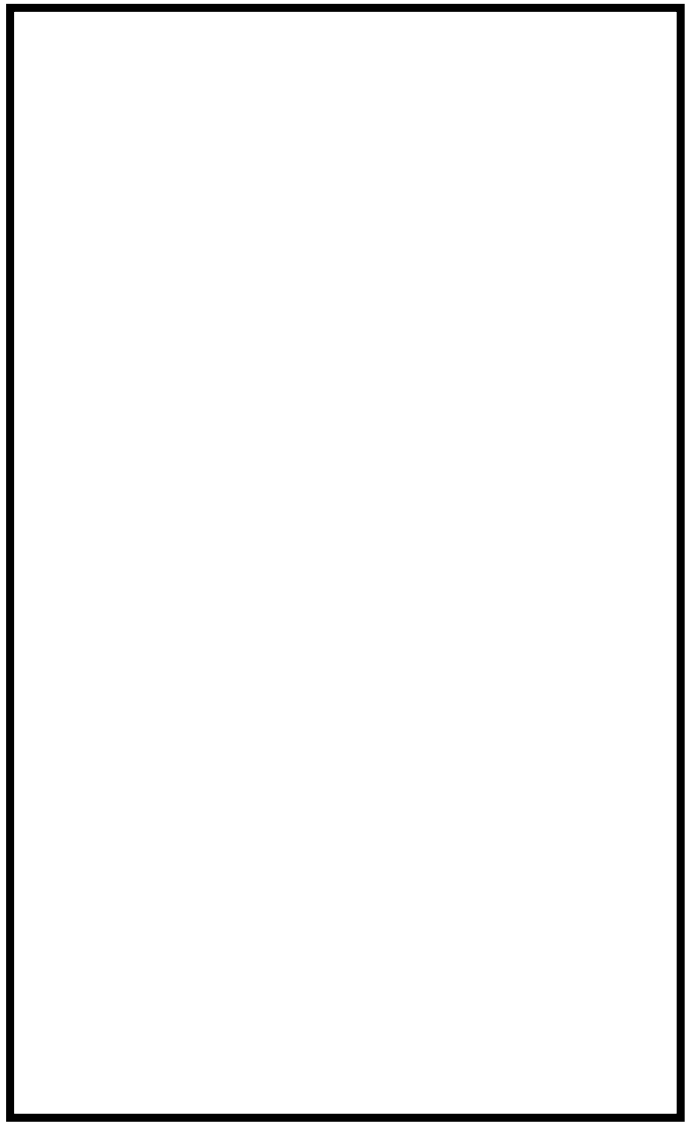
```
public void affiche( )
{









}
```

Mémoire de travail









UCT

Mémoire de programme **Mémoire globale**

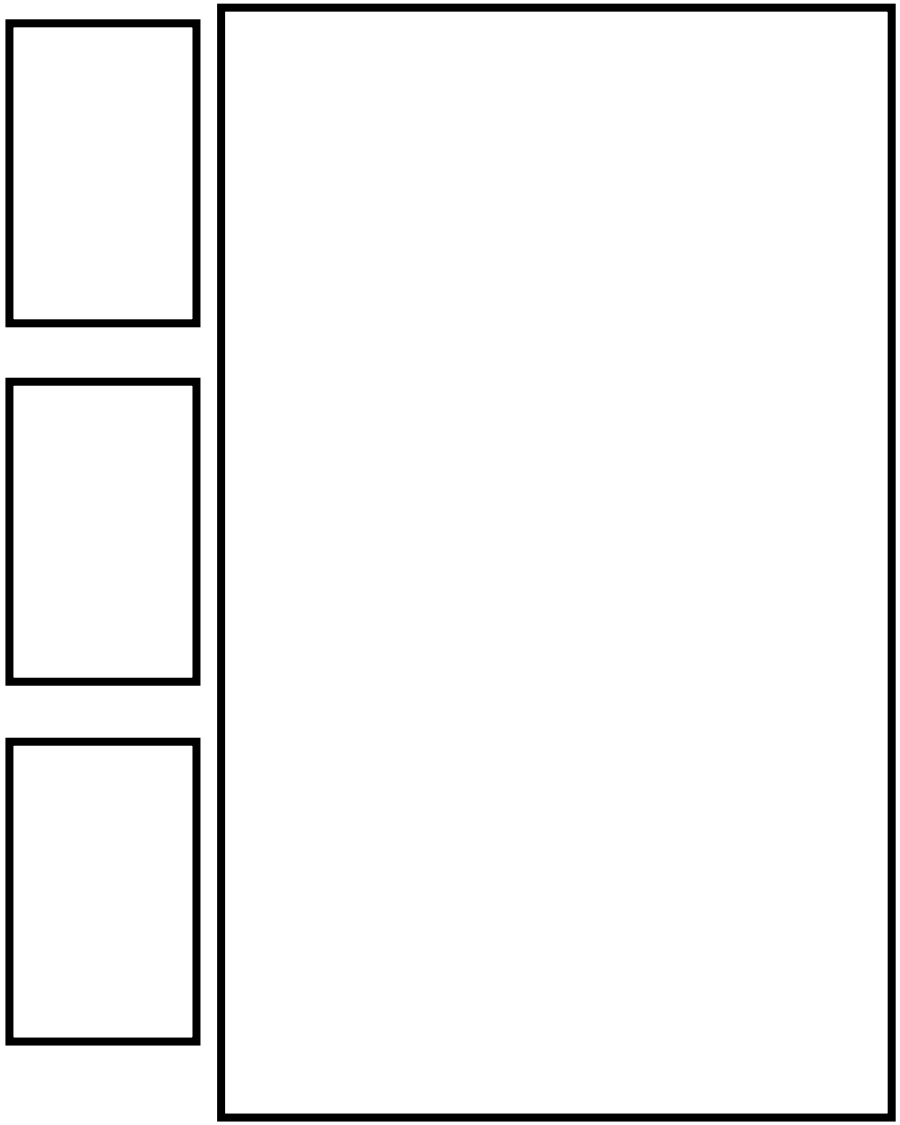
Exercice 11-10 – La somme de Fractions

```
public Fraction plus(Fraction opérande)
{




   return new Fraction(n, d); // Réduit automat.!
}
```

Exercice 11-11: Ajouter un entier à une fraction

```
public Fraction plus(int entier)
{




}
```

Mémoire de travail



Mémoire de programme **Exercice- L'utilisation de Fraction** **Mémoire globale**

```
// Cette classe teste notre réalisation de la classe Fraction
public class TestFraction
{
  public static void main(String [ ] args)
  {
    // teste les constructeurs et l'affichage
    Fraction a = new Fraction (1, -4);
    Fraction b = new Fraction (-14, -24);
    Fraction c = new Fraction (0, -3);
    Fraction d = new Fraction (2);
    Fraction e = new Fraction (0);
    int i = 2;
    a.affiche( ); System.out.println( ); //-1/4
    b.affiche( ); System.out.println( ); //7/12
    c.affiche( ); System.out.println( ); //0
    d.affiche( ); System.out.println( ); //2
    e.affiche( ); System.out.println( ); //0
    // teste somme
    a.plus(b).affiche( ); System.out.println(); //1/3
    a.plus(d).affiche( ); System.out.println(); //7/4
    c.plus(d).affiche( ); System.out.println(); //2
    e=Fraction.somme(-1, a.plus(i));
    e.affiche( ); System.out.println( );       //3/4
  }
}
```

Mémoire de travail

Fenêtre console

UCT