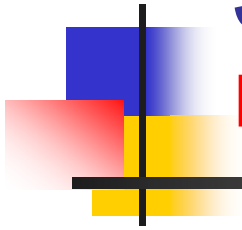
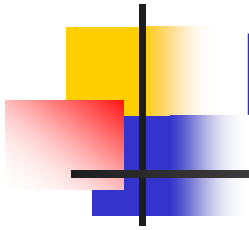


Java

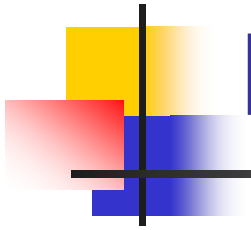
Licence Professionnelle CISII, 2009-10

Cours 4 : Programmation structurée



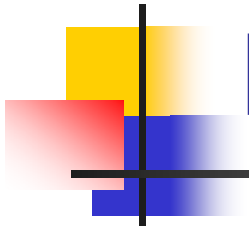
Programmation structurée

- Principe
  - Les méthodes sont structurées en blocs par les structures de la programmation structurée
    - suites de blocs
    - alternatives
    - répétitions
  - Un bloc est un ensemble d'instructions délimité par {}
  - Les blocs peuvent être emboîtés les uns dans les autres



Programmation structurée

- Portée des identificateurs
  - Les blocs définissent la portée des identificateurs
  - La portée d'un identificateur commence à l'endroit où il est déclaré et va jusqu'à la fin du bloc dans lequel il est défini, y compris dans les blocs emboîtés



Programmation structurée

- Déclaration des variables locales- compléments
  - Les variables locales peuvent être déclarées n'importe où dans un bloc (pas seulement au début)
  - On peut aussi déclarer la variable qui contrôle une boucle « **for** » dans l'instruction « **for** » (la portée est la boucle) :

```
for (int i = 0; i < 8; i++) {
        s += valeur[i];
}
```

Programmation structurée

- Portée des identificateurs : Portee1.java
  - Attention ! Java n'autorise pas la déclaration d'une variable dans un bloc avec le même nom qu'une variable d'un bloc emboîtant, ou qu'un paramètre de la méthode

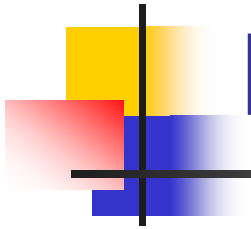
```
int somme(int init) {
   int i = init;
   int j = 0;
   for (int i=0; i<10; i++) { //duplication de variable locale
         j += i;
}
int init = 3; { //duplication de variable locale
```

Programmation structurée

- Instructions **de contrôle**
  - Alternative « **if** ...**else** »

if (*expressionBooléenne***)**
bloc-instructions ou instruction
else
bloc-instructions ou instruction

```
int x = y + 5;
if (x % 2 == 0) {
        type = 0;
        x++;
}
else
        type = 1;
```



Programmation structurée

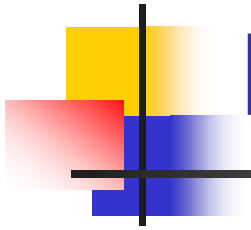
- Expression conditionnelle

expressionBooléenne ? *expression1* : *expression2*

```
int y = (x % 2 == 0) ? x + 1 : x;
```

est équivalent à

```
int y;
if (x % 2 == 0)
y = x + 1
else
y = x;
```

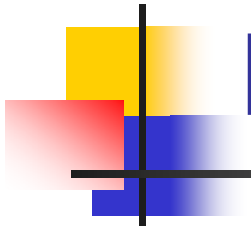


Programmation structurée

- Distinction de cas suivant une valeur

```
switch(expression) {
case val1: instructions;
break;
...
case valn: instructions;
break;
default: instructions;
}
```

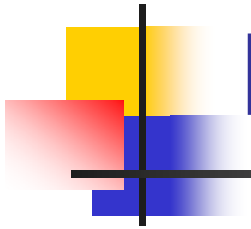
  - Sans break les instructions du cas suivant sont exécutées !
  - *expression* est de type **char**, **byte**, **short**, ou **int**, ou de type énumération
  - S'il n'y a pas de clause **default**, rien n'est exécuté si *expression* ne correspond à aucun **case**



Programmation structurée

- Exemple de **switch**

```
int nbVoyelles = 0, nbA = 0,
nbT = 0, nbAutre = 0;
. . .
switch (lettre) {
    case 'a' : nbA++;
    case 'e' : // pas d'instruction !
    case 'i' : nbVoyelles++;
    break;
    case 't' : nbT++;
    break;
    default : nbAutre++;
}
```



Programmation structurée

- Répétitions « tant que »

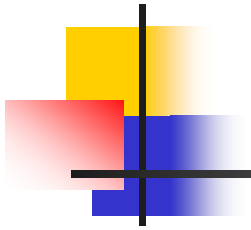
  while(*expressionBooléenne***)**

  bloc-instructions ou instruction

  - **do**

  bloc-instructions ou instruction

  while(*expressionBooléenne***)**



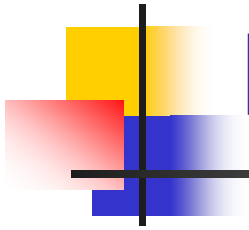
Programmation structurée

Répétition **for**

for(*init***;** *test***;** *incrément***){**
  *instructions***;**
}

- est équivalent à

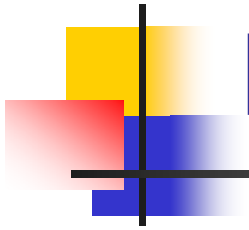
init;
while (*test*) {
  nstructions;
  Incrément
}



Programmation structurée

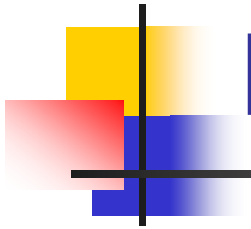
Exemple de **for**

```
int somme = 0;
for (int i=0; i < tab.length; i++) {
  somme += tab[i];
}
System.out.println(somme);
```



Programmation structurée

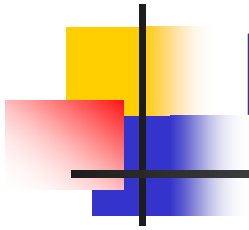
- « for each »
  - Une nouvelle syntaxe introduite par la version 5 du JDK simplifie le parcours d'un tableau
  - La syntaxe est plus simple/lisible qu'une boucle *for* ordinaire
  - Attention, on ne dispose pas de la position dans le tableau (pas de « variable de boucle »)
  - On verra par la suite que cette syntaxe est encore plus utile pour le parcours d'une « collection »



Programmation structurée

Parcours d'un tableau

```
String[] noms = new String[50];
...
// Lire « pour chaque nom dans noms »
// « : » se lit « dans »
for (String nom : noms) {
  System.out.println(nom);
}
```

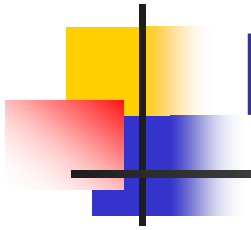


Programmation structurée

Parcours d'un tableau

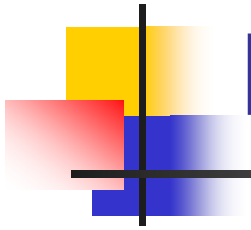
Exemple

```
public class ForeachArray {
    public static void main(String args[]) {
      String[] data = { "Toronto", "Stockholm" };
      for (String s : data) {
        System.out.println(s);
      }
    }
  }
```



Programmation structurée

- Instructions liées aux boucles
  - **break**
    - sort de la boucle et continue après la boucle
  - **continue**
    - passe à l'itération suivante
  - **break** et **continue**
    - peuvent être suivis d'un nom d'étiquette qui désigne une boucle englobant la boucle où elles se trouvent (une étiquette ne peut se trouver que devant une boucle)



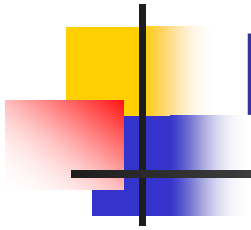
Programmation structurée

- Exemple de **continue** et **break**

```
int somme = 0;
for (int i=0; i < tab.length; i++) {
   if (tab[i] == 0) break;
   if (tab[i] < 0) continue;
   somme += tab[i];
}
System.out.println(somme);
```

  - Qu'affiche ce code avec le tableau

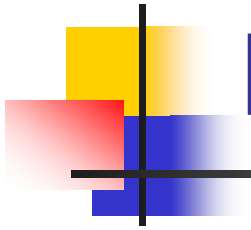
    1 ; –2 ; 5 ; -1 ; 0 ; 8 ; -3 ; 10 ?



Programmation structurée

- Étiquette de boucles

```
boucleWhile:
while (pasFini) {
...
for (int i=0; i < t.length; i++) {
    ...
    if (t[i] < 0)
    continue boucleWhile;
    ...
}
...
}
```



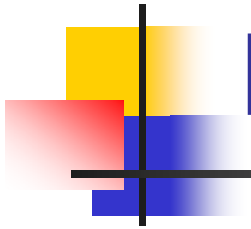
Programmation structurée

- Passage des arguments des méthodes
  - Le passage se fait par valeur (les valeurs des arguments sont recopiées dans l'espace mémoire de la méthode)
  - Attention, pour les objets, la valeur passée est une référence ; donc,
    - si la méthode modifie l'objet référencé par un paramètre, l'objet passé en argument sera modifié en dehors de la méthode
    - si la méthode change la valeur d'un paramètre (type primitif ou référence), ça n'a pas d'incidence en dehors de la méthode

- Exemple de passage par valeur

```
public class ParValeur{
public static void main (String [] paramètre)   {
   // Déclaration des variables
   int valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant tripler() ");
   tripler(valeur);
   System.out.println("Valeur = " + valeur + " apres tripler() ");
} // fin de main()

  public static void tripler (int valeur)  {
   System.out.println("Valeur = " + valeur + " dans tripler() ");
    valeur = 3 * valeur;
   System.out.println("Valeur = " + valeur + " dans tripler() ");
  } // fin de tripler
} //fin de class ParValeur
```



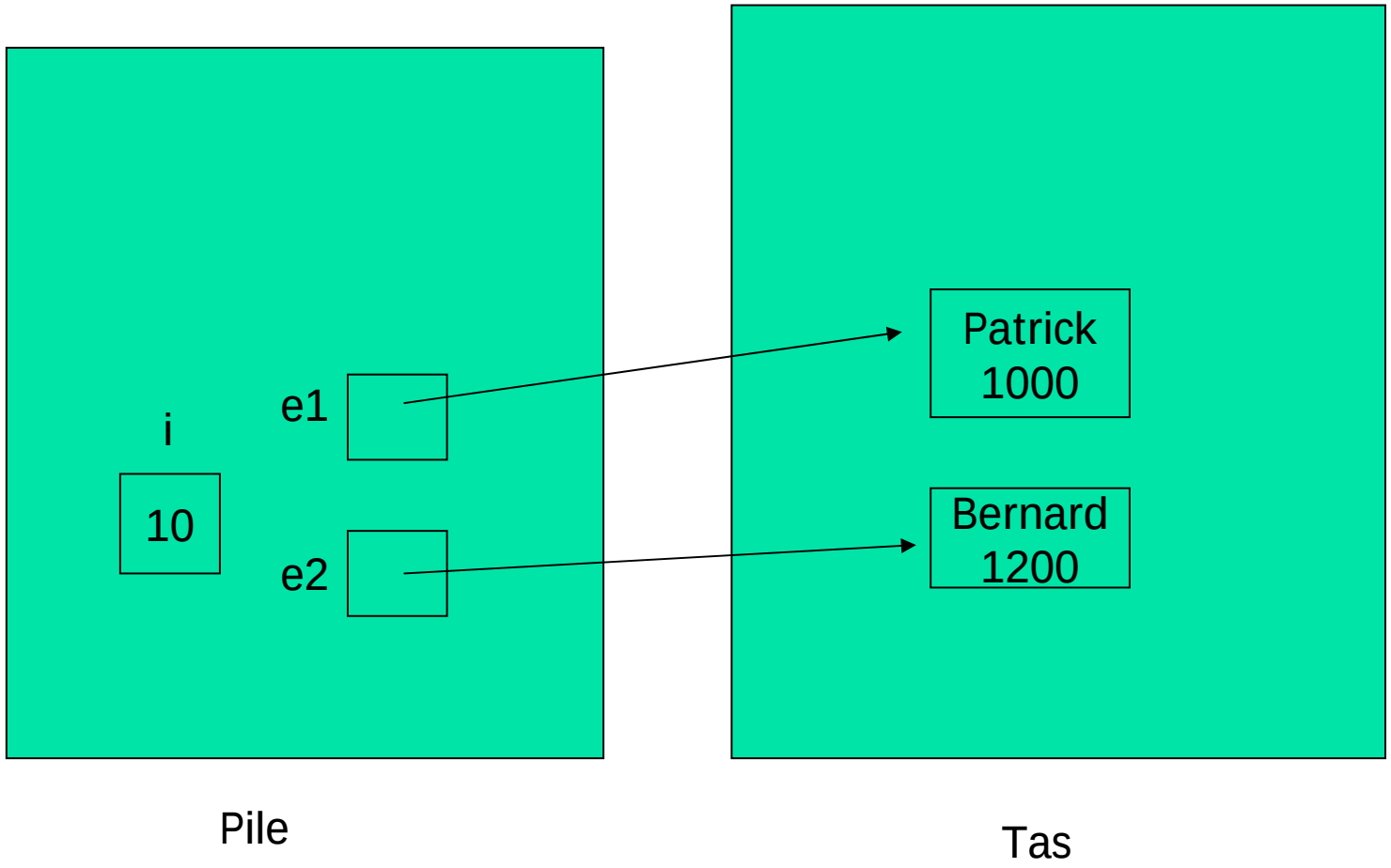
Programmation structurée

- Exemple de passage de paramètres

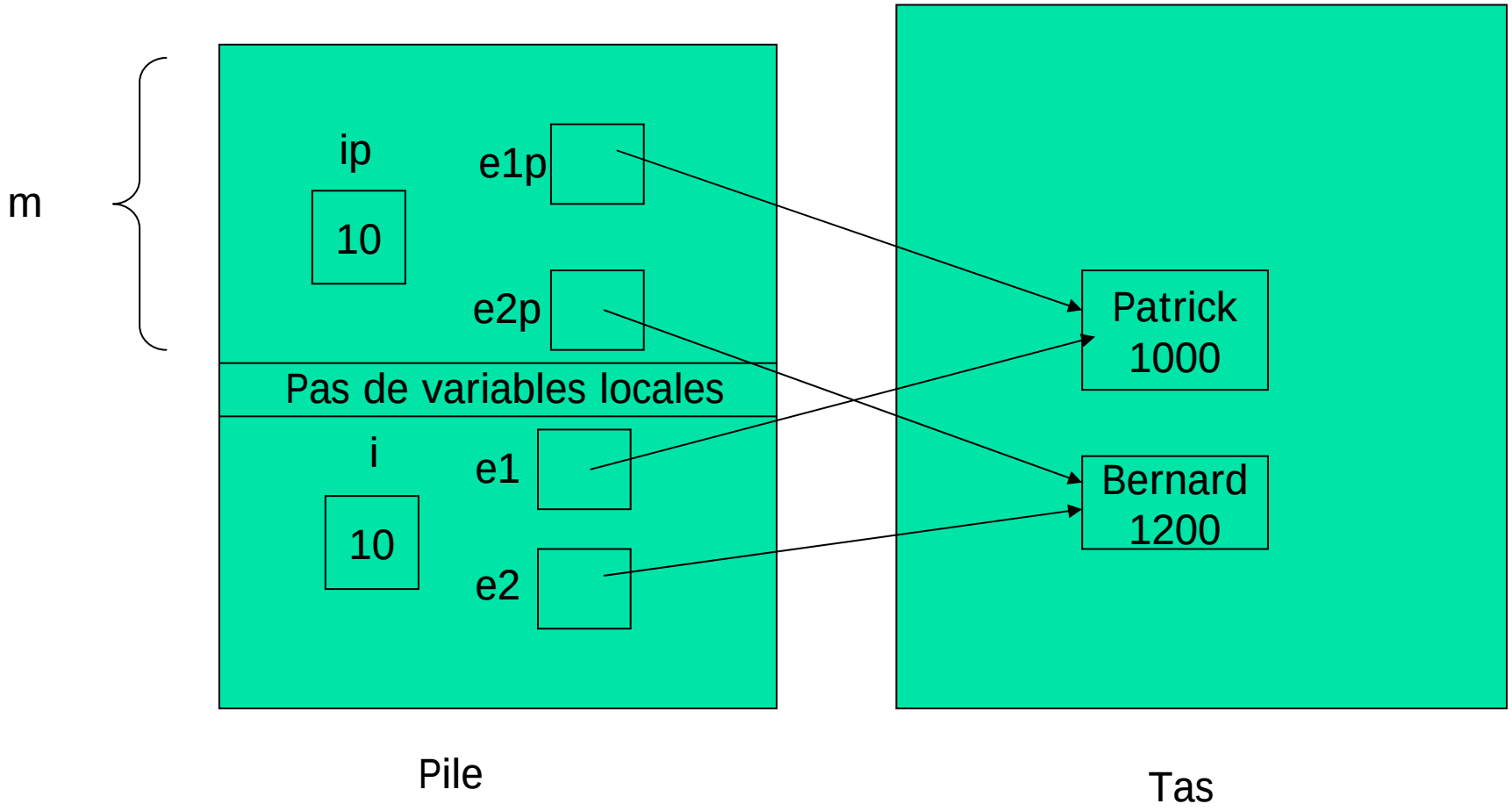
```
public static void m(int ip, Employe e1p, Employe e2p) {
   ip = 100;
   e1p.salaire = 800;
   e2p = new Employe("Pierre", 900);
}
public static void main(String[] args) {
   Employe e1 = new Employe("Patrick", 1000);
   Employe e2 = new Employe("Bernard", 1200);
   int i = 10;
   m(i, e1, e2);
   System.out.println(i + '\n' + e1.salaire+ '\n' + e2.nom);
}
```

- Que sera-t-il affiché ?
  - **10 800.0 Bernard**

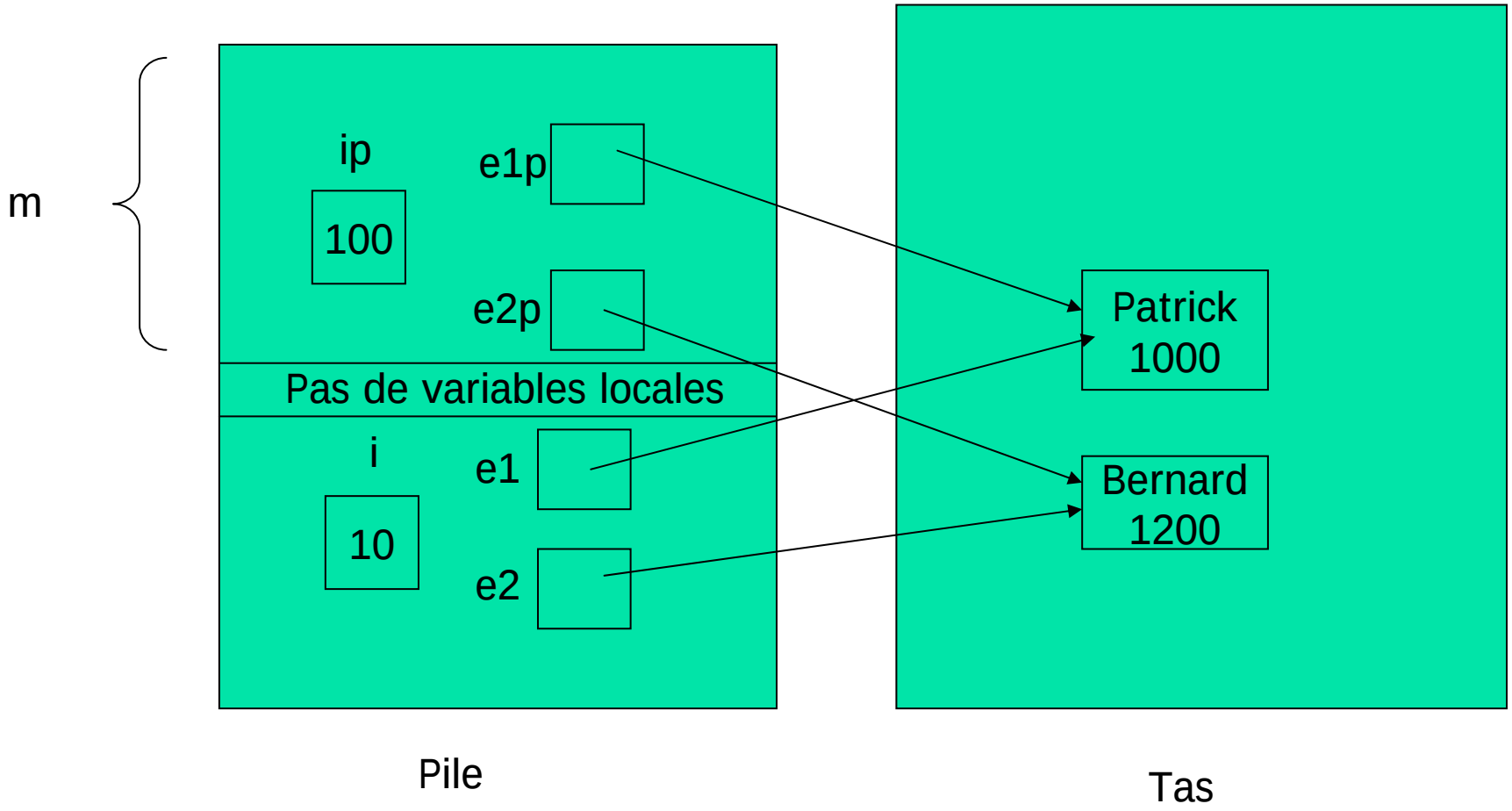
```
main()
Employe e1 = new Employe("Patrick", 1000);
Employe e2 = new Employe("Bernard", 1200);
int i = 10;
```



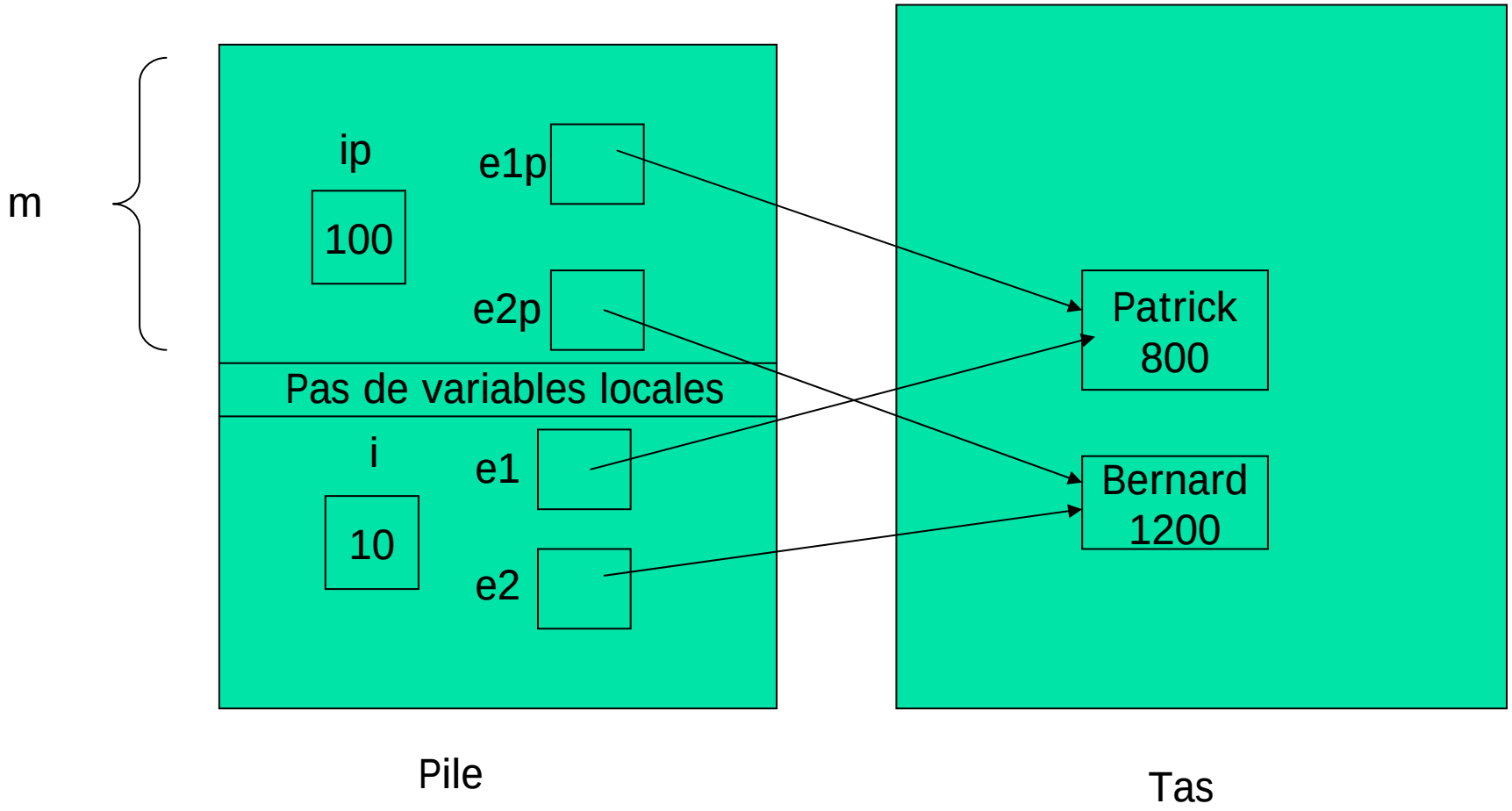
main():
m(i,e1,e2)


m
ip
10
e1p
e2p
Pas de variables locales
i
10
e1
e2
Patrick
1000
Bernard
1200
Pile
Tas

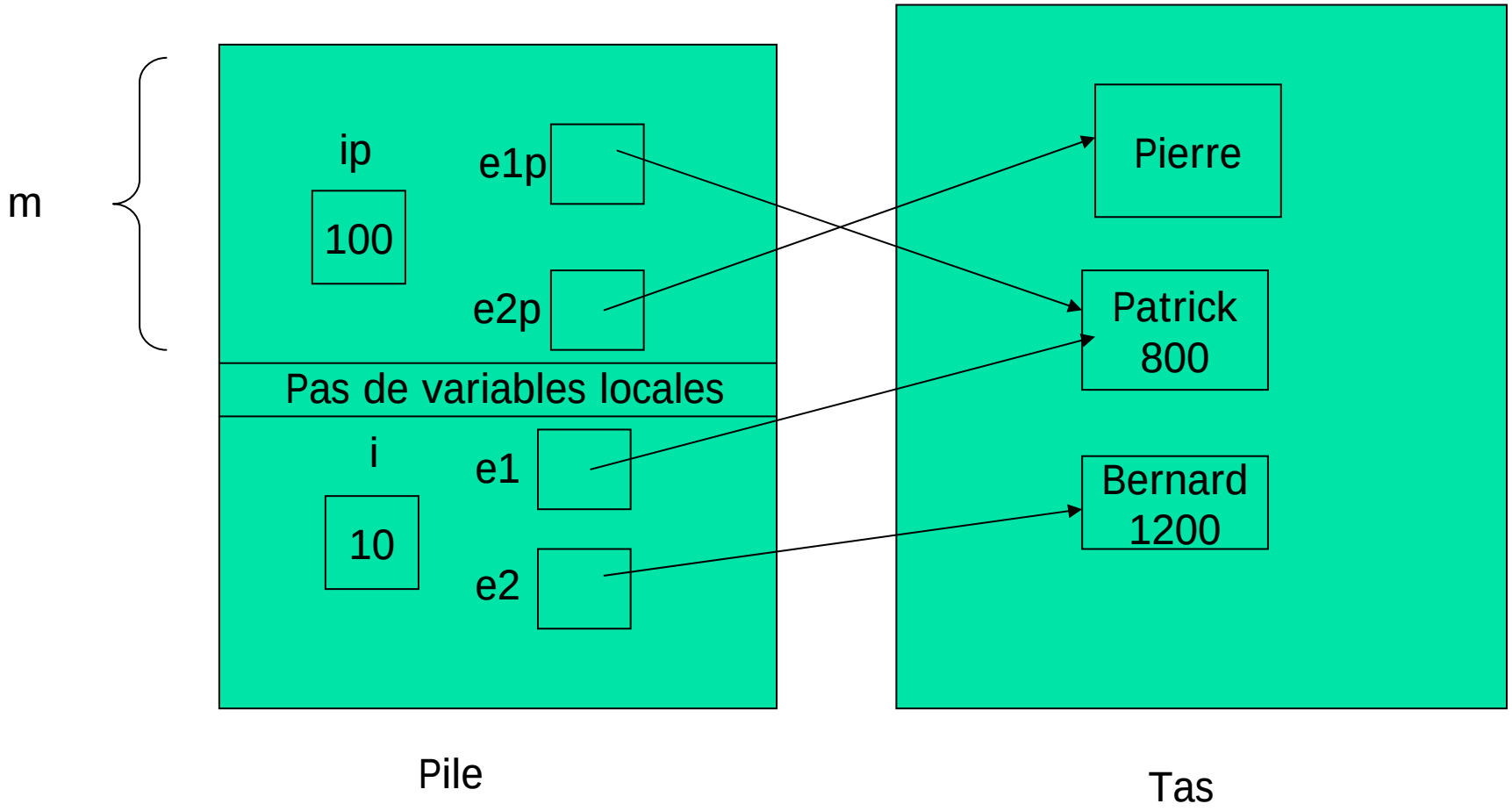
```
m():
ip = 100;
e1p.salaire = 800;
e2p = new Employe("Pierre", 900);
```



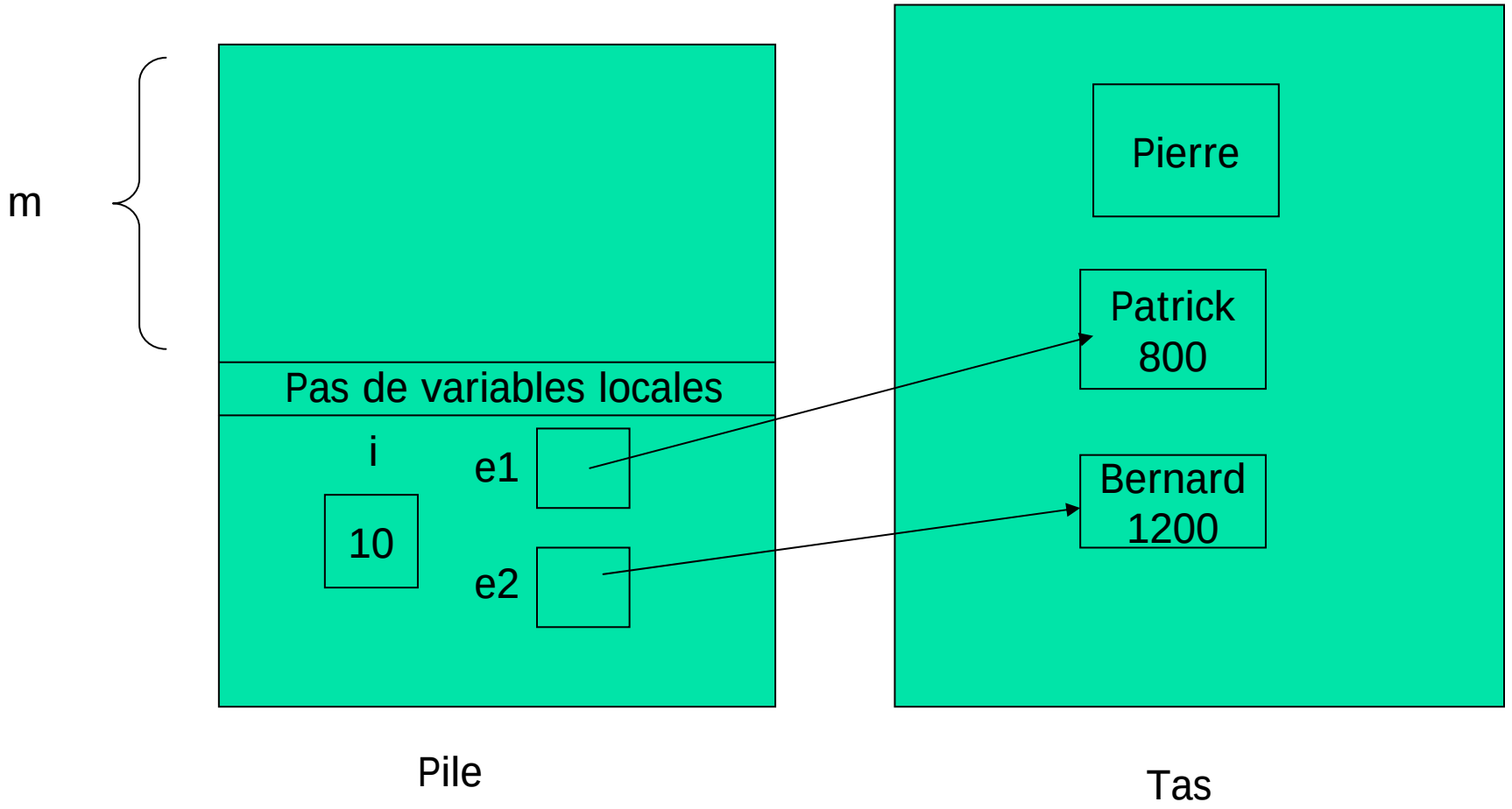
```
m():
ip = 100;
e1p.salaire = 800;
e2p = new Employe("Pierre", 900);
```

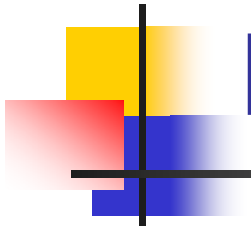


```
m():
ip = 100;
e1p.salaire = 800;
e2p = new Employe("Pierre", 900);
```



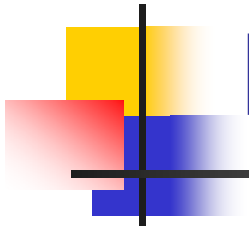
main():
System.out.println(i + '\n' + e1.salaire+ '\n' + e2.nom);





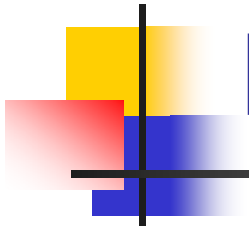
Programmation structurée

- Nombre variable d'arguments
  - Quelquefois il peut être commode d'écrire une méthode avec un nombre variable d'arguments
  - L'exemple typique est la méthode *printf* du langage C qui affiche des arguments selon un format d'affichage donné en premier argument
  - Depuis le JDK 5.0, c'est possible en Java



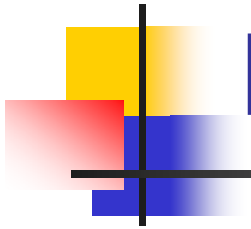
Programmation structurée

- Syntaxe pour arguments variables
  - A la suite du type du dernier paramètre on peut mettre « ... » :
  - Exemple
    - String...
    - Object...
    - int...



Programmation structurée

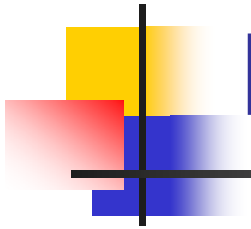
- Traduction du compilateur
  - Le compilateur traduit ce type spécial par un type tableau i.e.
    - **m(int p1, String... params)** est traduit par
    - m(int **p1, String[] params)**
  - Le code de la méthode peut utiliser **params** comme si c'était un tableau (boucle **for**, affectation, etc.)



Programmation structurée

Exemple

```
public static int max(int... valeurs) {
  int max = Integer.MIN_VALUE;
  for (int i : valeurs) {
  if (i > max)
  max = i;
  }
  return max;
}
```



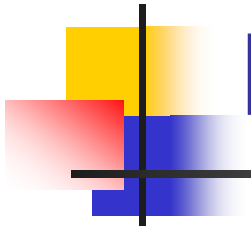
Programmation structurée

- Retour de la valeur d'une méthode
  - **return** sert à sortir d'une méthode en renvoyant une valeur (du type déclaré pour le type retour dans la définition de la méthode) :

```
return i * j;
return new Cercle(p, x+y);
```

  - **return** sert aussi à sortir d'une méthode sans renvoyer de valeur (méthode ayant **void** comme type retour) :

```
if (x == 0)
return;
```



Programmation structurée

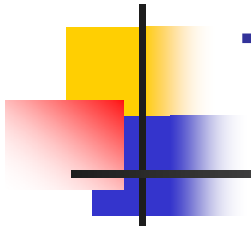
- Récursivité des méthodes
  - Les méthodes sont récursives ; elles peuvent s'appeler elles-mêmes :

```
static long factorielle(int n) {
  if (n == 0)
        return 1;
  else
        return n * factorielle(n - 1);
}
```

Que fait le programme suivant ?

```
public class ListeNoeud {
	private Integer element;
	private ListeNoeud suivant;
	public ListeNoeud(Integer i, ListeNoeud l) {
		element = i;
		suivant = l;
	}
	public ListeNoeud(Integer i) {
		this(i,null);
	}
	public ListeNoeud getSuivant() {
		return suivant;
	}
	public String toString() {
		return element.toString();
	}
}
public class Liste {
	private ListeNoeud premier;
	public Liste() {
		premier = null;
	}
	public boolean estVide() {
		return (premier == null);
	}
	public void inserer(Integer i) {
		if (estVide())
		premier = new ListeNoeud(i);
		else
		premier = new ListeNoeud(i,premier);
	}
	public String toString() {
		String s = "";
		if (estVide())
		return "Vide";
		ListeNoeud p = premier;
		while (p != null) {
			s = s + " " + p.toString();
			p = p.getSuivant();
		}
		return s;
}

public static void main (String [] args) {
	Liste l = new Liste();
	l.inserer(new Integer(1));
	l.inserer(new Integer(2));
	l.inserer(new Integer(3));
	System.out.println(l);
	}
}
```



TD 4

- Exercices 1-3