M2202 - Algorithmique

Philippe Gambette
Thomas Guérin, Thomas Bardoux

Page web du cours : http://tinyurl.com/M2202-2015

- **Contact**

- Courriel : philippe.gambette@gmail.com (M2202 doit apparaître dans le sujet du courriel)
- Avant ou après le cours
- Possibilité de poser des questions, de demander des exercices supplémentaires d'entrainement.

- **Notes et devoirs**

Chez soi :

- exercices sur elearning pour vous inciter à travailler **régulièrement** et corriger vos erreurs. **Annonce sur elearning 6 jours avant**.

Pendant les cours :

- Note globale de TP à la fin du semestre, prenant en compte votre avancée à chaque séance et/ou note de remplissage de cours à trous.
- Devoir final le 9 juin : documents interdits sauf deux feuilles recto-verso.

- **Sources**
  - Cours de Jean-François Berdjugin à l'IUT de Grenoble http://berdjugin.com/enseignements/inf/inf220/
  - http://xkcd.com, http://www.xkcd.fr
  - Cours de Xavier Heurtebise à l'IUT de Provence http://x.heurtebise.free.fr
  - *Le livre de Java premier langage*, d'A. Tasso

Récursivité

2 façons de le voir :

- Une mise en abyme

Comment dessiner ces images ?
Pour dessiner la grande image, j'utilise le dessin de l'image en plus petit.

Qu'est-ce qu'une poupée russe ?

- Une baguette magique algorithmique

Si je sais construire le *n*-ième objet à partir du (*n*-1)-ième objet, et que je sais construire le premier objet, alors je sais construire tous les objets.

La "minute xkcd" - Jeu de rôle sur table

http://xkcd.com/244
http://xkcd.free.fr?id=244

Exemple des factorielles :
factorielle(1) = 1
factorielle(2) = 1x2 = 2
factorielle(3) = 1x2x3 = 6
factorielle(4) = 1x2x3x4 = 24

Initialisation :
factorielle(1) =
Formule d'hérédité :
factorielle(*n*) =

Photo Ethan Clements http://ethanclements.blogspot.com/2010/07/mise-en-abyme.html
Vache qui rit : http://www.apprendre-en-ligne.net/blog/index.php/2008/03/29/916-mise-en-abyme

Algorithme récursif de calcul de la factorielle

Exemple des factorielles :

- Initialisation : la première factorielle est 1
- Hérédité : pour passer de factorielle(*n*-1) à factorielle(*n*), je **multiplie par *n***

Algorithme **factorielle**
Entrées :
Type de sortie :
Variable :
Début

Fin

Trace de **factorielle**(4) :

La "minute mathématique"
Théorème : Je sais monter une échelle
Démonstration :
Initialisation : je sais monter depuis le sol jusqu'au premier barreau.
Hérédité : si je sais monter jusqu'au (*n*-1)-ième barreau, je saurai monter jusqu'au *n*-ième barreau.

Algorithme récursif de calcul du produit

Exemple du produit :

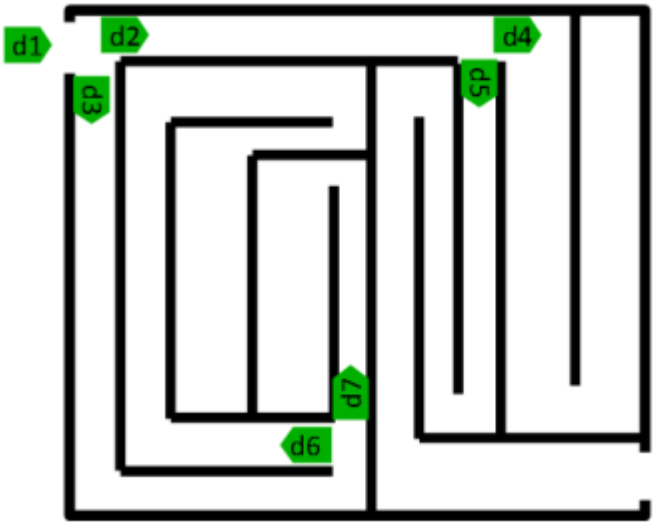
- Initialisation : $a \times 0 = 0$
- Formule d'hérédité : $a \times n = a \times (n-1) + a$

Algorithme **produit**
Entrées :
Type de sortie :
Variable :
Début

Fin

Trace de **produit**(4,3) :

Algorithme récursif de parcours groupé d'un labyrinthe



Le labyrinthe et les robots tueurs :
10 personnes dans un labyrinthe poursuivies par des robots tueurs. Si on appuie sur le bouton stop dans les 5 minutes, les robots sont désactivés.

Algorithme **trouveBouton**
Entrées : entier *nombrePersonnes*, flottant *tempsRestant*, chaîne de caractères *direction*
Type de sortie : booléen

Si on trouve le bouton à temps, on appuie dessus, et l'algorithme renvoie VRAI. Si on trouve une intersection, le groupe se sépare en deux. Si on ne trouve pas le bouton à temps, on renvoie FAUX.

Que renvoie **trouveBouton**(10,5,"d1") ?

Arbre d'exécution de **trouveBouton**(10,5,"d1").

trouveBouton(10,5,"d1")

Principe d'un algorithme de tri

Entrée :
Ensemble d'éléments **tous comparables deux à deux** par un ordre **≤**

Sortie :
Éléments **triés selon cet ordre**, du plus petit au plus grand

Ensemble d'éléments	**Ordre**	**Tous comparables deux à deux ?**
tableau d'entiers	plus petit	
tableau de chaînes de caractères	lexicographique (du dictionnaire)	
tableau de cartes à jouer	2 ≤ 3 ≤ … ≤ 10 ≤ V ≤ D ≤ R ≤ 1	
tableau de cartes à jouer	♣ ≤ ♦ ≤ ♥ ≤ ♠ puis 2 ≤ 3 ≤ … ≤ 10 ≤ V ≤ D ≤ R ≤ 1	

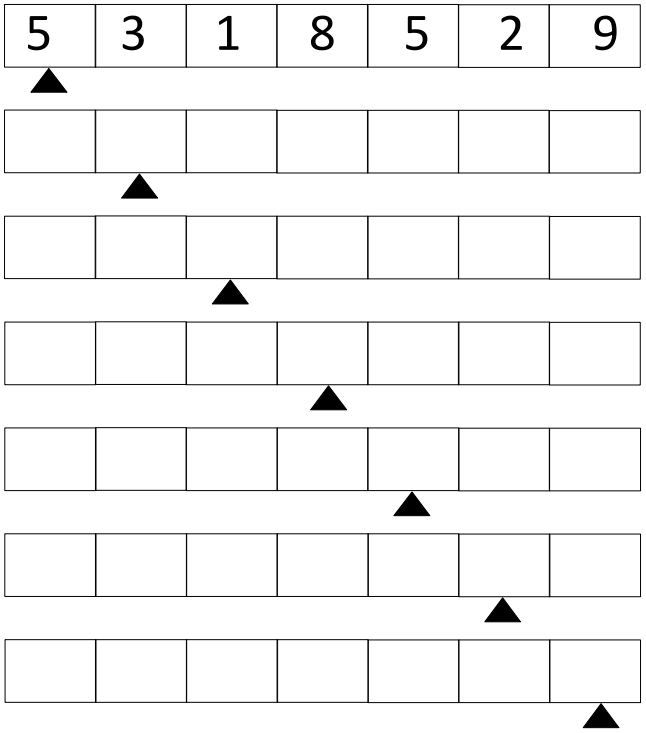
Exemple avec un tableau d'entiers :

en anglais : trié = sorted / trier = to sort

Tri par sélection (ou tri par extraction)

Idée : à la *i*-ième étape, on prend le plus petit élément à partir de la *i*-ième case (comprise) et on l'échange avec l'élément de la *i*-ième case.

Exemple avec un tableau d'entiers :




```
Algorithme positionMinimum
Entrée : tableau d'entiers tab,
entier debut
Type de sortie : entier
Variables : entiers position, i, min
Début
  position ←
  i ←
  min ←
  Tant que i≤Longueur(tab) faire :
        Si                          alors :
          position ← i
          min ← Case(tab,i)
        Fin Si
        i ← i+1
  Fin Tant que
  renvoyer position
Fin
```

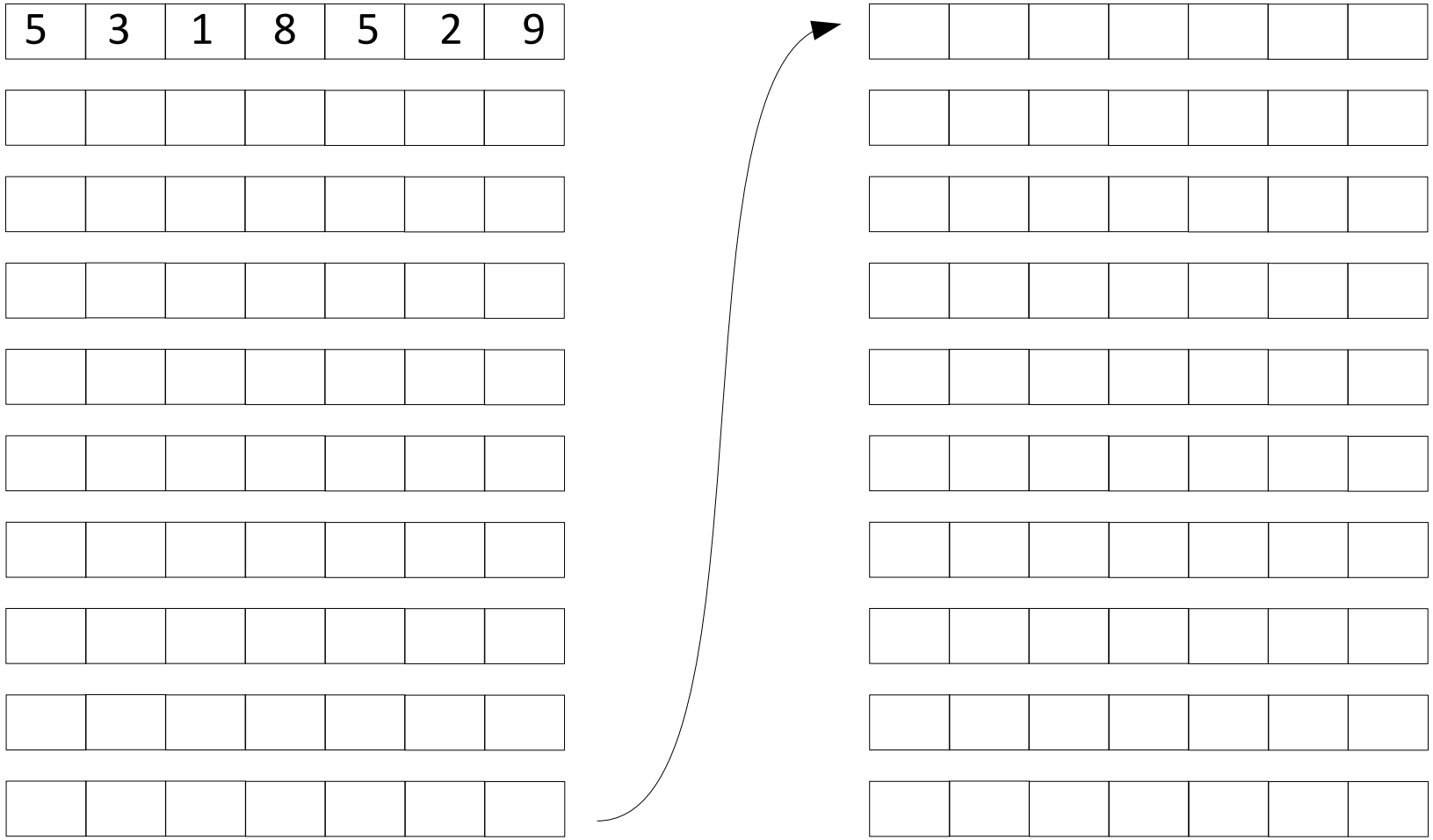
```
Algorithme triSelection
Entrée : tableau d'entiers tab
Type de sortie : tableau d'entiers
Variables : entiers i, temp, posMin
Début
  i ←
  Tant que i<Longueur(tab) faire :
     posMin ← positionMinimum(tab,i)
     //posMin peut être égal à i
     temp ← Case(tab,i)
     Case(tab,i) ← Case(tab,posMin)
     Case(tab,posMin) ← temp
     i ← i+1
  Fin Tant que
  renvoyer tab
Fin
```

Tri **en place** :
on arrive à trier le tableau sans avoir besoin de créer un nouveau tableau.

Tri à bulles

Idée : tant que le tableau n'est pas trié, on le parcourt en effectuant l'opération suivante à la *i*-ième case : si elle est supérieure à la suivante, on les échange.

Exemple avec un tableau d'entiers :



Les listes

Le **problème** des **tableaux** :
- taille fixe
- insertion “difficile” d'un élément
- suppression “difficile” d'un élément

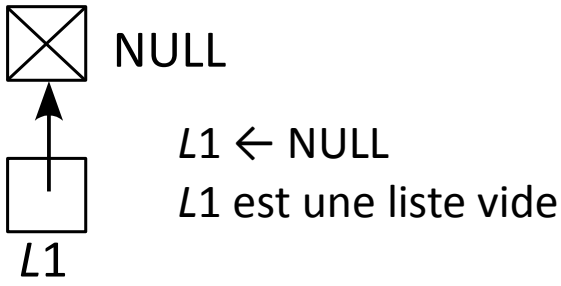
L'intérêt des **listes** :
- taille variable
- insertion facile d'un élément
- suppression facile d'un élément

Mais :
- accès “difficile” au *i*-ième élément

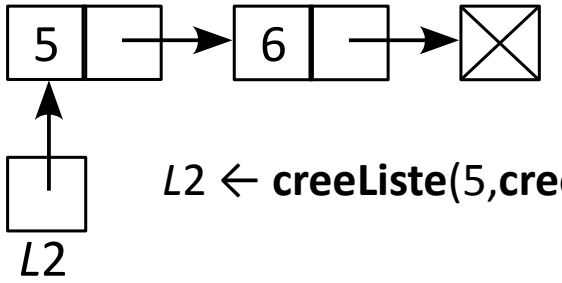
Une **liste “simplement chaînée”** est :
- soit une liste vide
- soit une première case qui contient une valeur, et qui pointe vers une liste

Une liste d'entiers est :
- soit une liste vide
- soit une première case qui contient un entier, et qui pointe vers une liste d'entiers

Liste “simplement chaînée” : la **chaîne d'une ancre**
Quand l'ancre est jetée et toute la chaîne déroulée, on a facilement accès au premier maillon (sur le bateau). Pour les suivants il faut remonter l'ancre. Quand on a un maillon sous les yeux on peut facilement le supprimer (le casser et accrocher le précédent au suivant), en insérer un...



*L*1 ← NULL
*L*1 est une liste vide



*L*2 ← **creeListe**(5,**creeListe**(6, NULL))

Fonctions de base sur les listes

Sur une liste *L*1 ou *L*2, on peut :

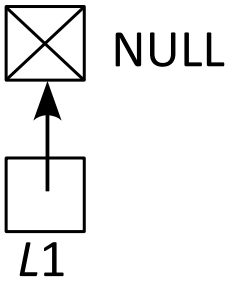
- savoir si elle est vide : *L*1 = NULL ?

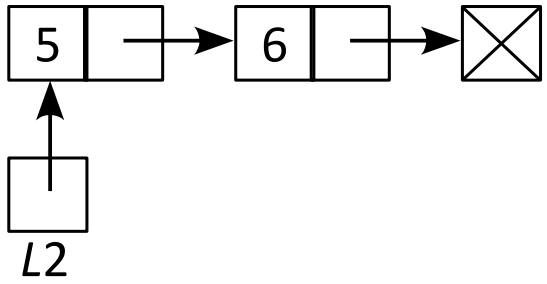
si *L*2 est non vide :

- accéder à la valeur de son premier élément : **tete**(*L*2) renvoie ...

si *L*2 est non vide :

- accéder à la liste qui commence à la case suivante : **suivant**(*L*2) renvoie ...





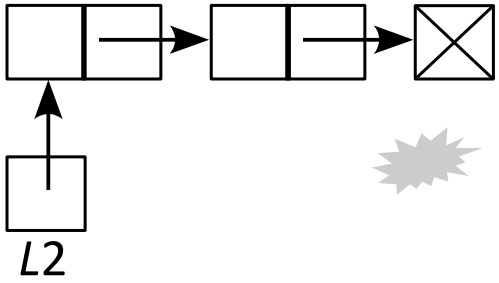
Modification de l'élément en tête d'une liste (facile pour liste, facile pour tableau)

Écrire l'algorithme **remplaceTete** qui prend en entrée une liste d'entiers *L* et un entier *a*, et renvoie la liste L dont le premier élément est remplacé par *a*

Algorithme **remplaceTete**
Entrées : liste d'entiers *L*, entier *a*
Type de sortie : liste d'entiers
Début

renvoyer
Fin

remplaceTete(*L*2,2) :



Fonctions de base sur les listes

Suppression de l'élément en tête d'une liste (facile pour liste, pas facile pour tableau)

Par exemple, supprimer 5 au début de la liste
L=**creeListe**(5,**creeListe**(6, NULL))

Insertion d'un élément en tête d'une liste (facile pour liste, pas facile pour tableau)

Par exemple, insérer 2 au début de la liste
L=**creeListe**(5,**creeListe**(6, NULL))

Fonctions de base sur les listes

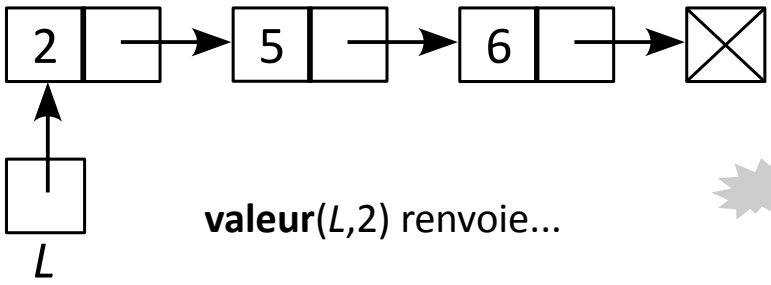
Valeur du *i*-ième élément de la liste (pas facile pour une liste, facile pour un tableau)

Nécessite de **parcourir la liste !**

Ecrire la fonction **valeur** qui prend en entrée une liste d'entiers *L* et un entier *i*, et qui renvoie la valeur de la *i*-ième case de *L*, si elle existe, et -1 sinon.

Algorithme **valeur**
Entrées : liste d'entiers *L*, entier *a*
Type de sortie :
Variables :
Début

Fin



valeur(*L*,2) renvoie...

Piles et files

Files
FIFO “First In First Out”

File d'attente à la Poste

Quatre fonctions :
- **créer** : créer une file vide
- **enfiler** : ajouter un élément en fin de file
- **défiler** : lire la valeur de l'élément au début de la file en l'enlevant de la file
- **tester si vide** : savoir si la file est vide

implémentable avec liste ***doublement*** chaînée

Piles
LIFO “Last In First Out”

Pile de documents à travailler

Quatre fonctions :
- **créer** : créer une pile vide
- **empiler** : ajouter un élément en haut de la pile
- **dépiler** : lire la valeur de l'élément en haut de la pile en l'enlevant de la pile
- **tester si vide** : savoir si la pile est vide

implémentable avec liste simplement chaînée

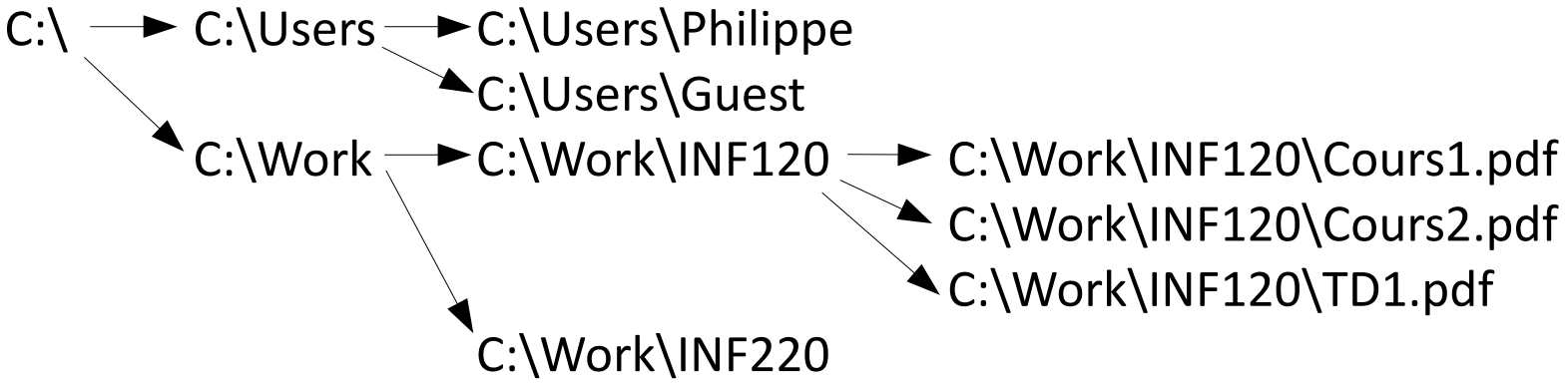
Pourquoi utiliser ces structures de données ?
- Plus simple à programmer qu'une liste
- Fonctions plus limitées, adaptées aux besoins

Intérêt des arbres

Intérêt des arbres :

• représentation d'un objet qui a une structure arborée :

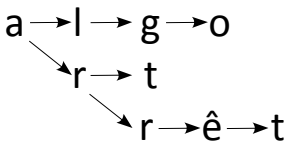
→ contenu d'un disque dur :



• stockage astucieux d'éléments

→ trouver rapidement le minimum des valeurs stockées

→ stocker un dictionnaire de manière compacte :

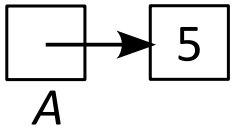


La "minute xkcd" - Arbre
http://xkcd.com/835
http://xkcd.free.fr/?id=835

Définition récursive et illustrée des arbres

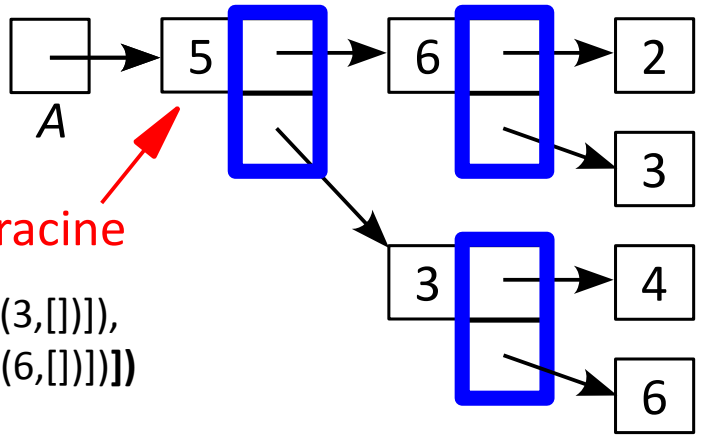
Un **arbre** :

- soit une **feuille**, qui contient une valeur, mais n'a pas d'enfant



$A \leftarrow$ **CreeArbre**(5,[])

- soit un **noeud** qui contient une valeur, et un tableau d'**enfants** dont chaque case pointe vers un **arbre**



$A \leftarrow$ **CreeArbre(**5,**[CreeArbre**(6,[**CreeArbre**(2,[]),**CreeArbre**(3,[])]),
CreeArbre(3,[**CreeArbre**(4,[]),**CreeArbre**(6,[])])**])**

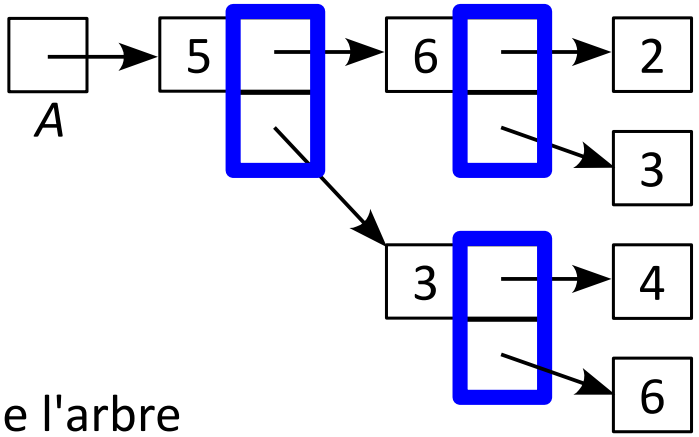
Si chaque tableau d'enfants a au plus deux cases, c'est un **arbre binaire**.

Fonctions de base sur les arbres

Tester si l'arbre est une feuille :
estFeuille(*A*)

Récupérer la valeur de la racine de l'arbre
racine(*A*)

Récupérer le tableau des enfants de la racine de l'arbre
enfants(*A*)



Concept de la programmation orientée objets

Objectif : modéliser des objets informatiques

- décrire leurs **caractéristiques**

- décrire leurs **comportements**

types d'objets Java que le programmeur peut définir

Avantages :

- code modulaire, décomposé en “classes” d'objets

- code sécurisé, possible de réutiliser certains objets dans d'autres programmes

→ simplifie la vie du programmeur

Exemples : - les actions Javascript sur les composants d'une page web

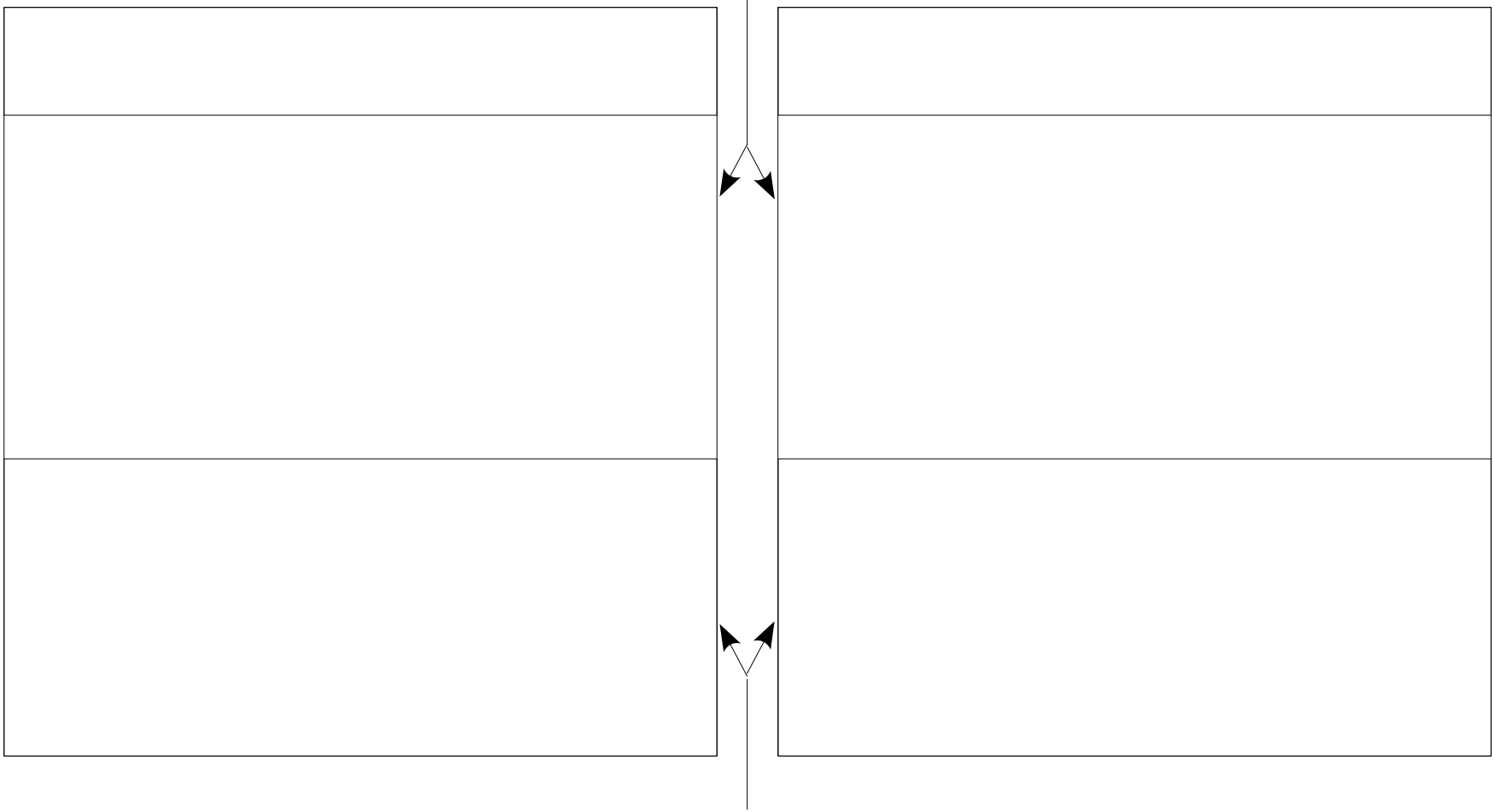
- la gestion des notes d'une classe

Modélisation objets

Quelles caractéristiques / attributs ? **“propriétés”**

Composants d'une page web

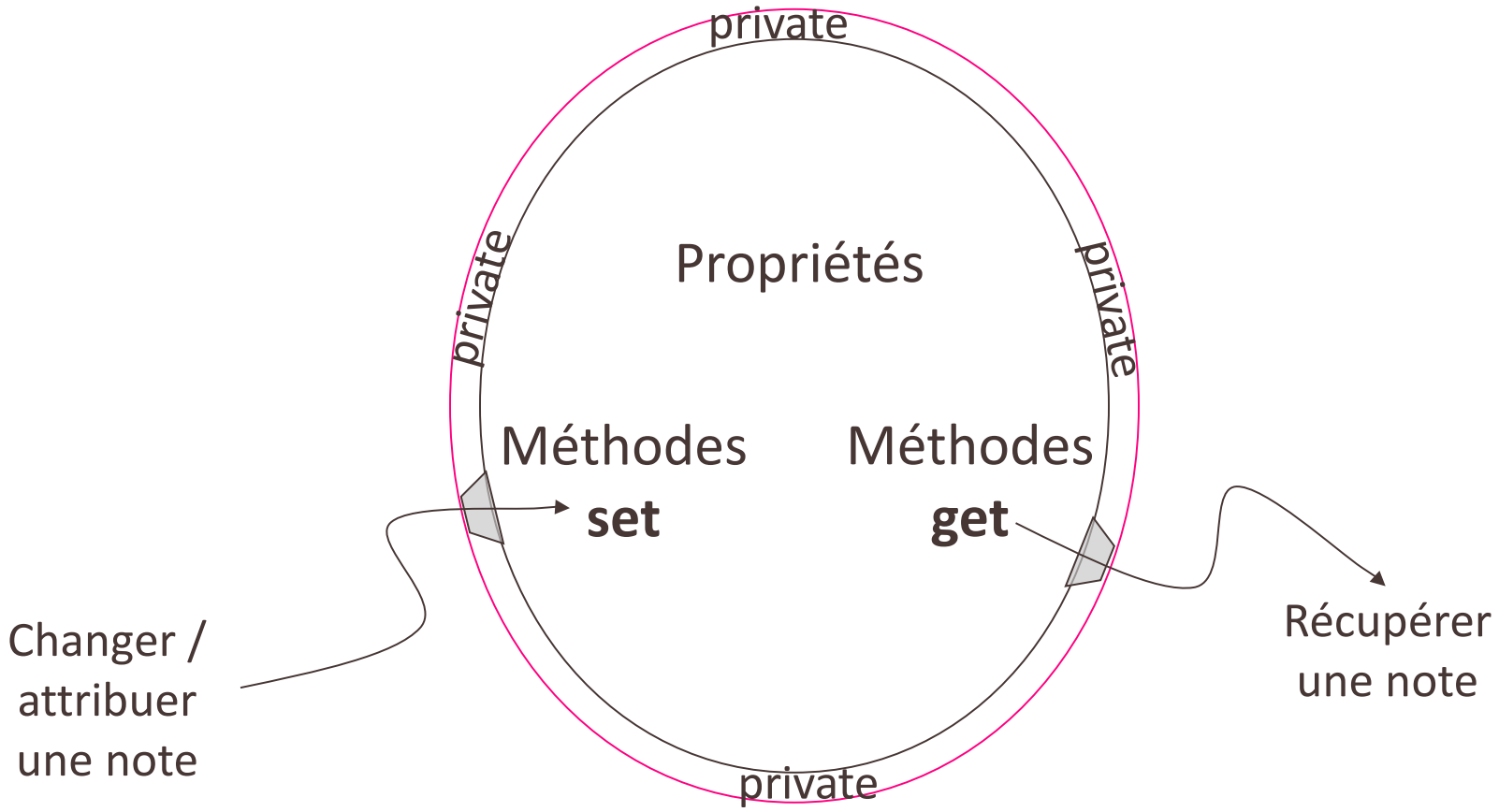
Etudiants d'une classe



Quels comportements / actions ? **“méthodes”**

Protection des données

Principe d'encapsulation



Protection des données

Constructeur

Sert à créer un objet

→ fournir une valeur à ses propriétés (initialisation)

Constructeur par défaut :

→ initialise les entiers à 1, les flottants à 0.0, les chaînes de caractères à Null.

Pour changer les valeurs des propriétés d'un objet par la suite :

- les modifier depuis une autre classe ?
- les modifier depuis une méthode de la classe de l'objet ?

Protection des données

Différents niveaux de protection (pour les propriétés et pour les méthodes)

→ public : accessible pour tous les objets de l'application

→ private : accessible que pour les méthodes de la même classe

→ protected : accessible que depuis les méthodes de la même classe ou d'une sous-classe

Partage d'une donnée entre objets d'une même classe :

→ static

→ sinon les données sont spécifique à un objet donné de la classe

Similaire à variable globale / variables locales, au niveau des objets plutôt que des fonctions

Syntaxe Java

Définir une nouvelle classe

```
public class Etudiant{




}
```

Syntaxe Java

Utiliser une classe depuis une autre (exemples de TP précédents)

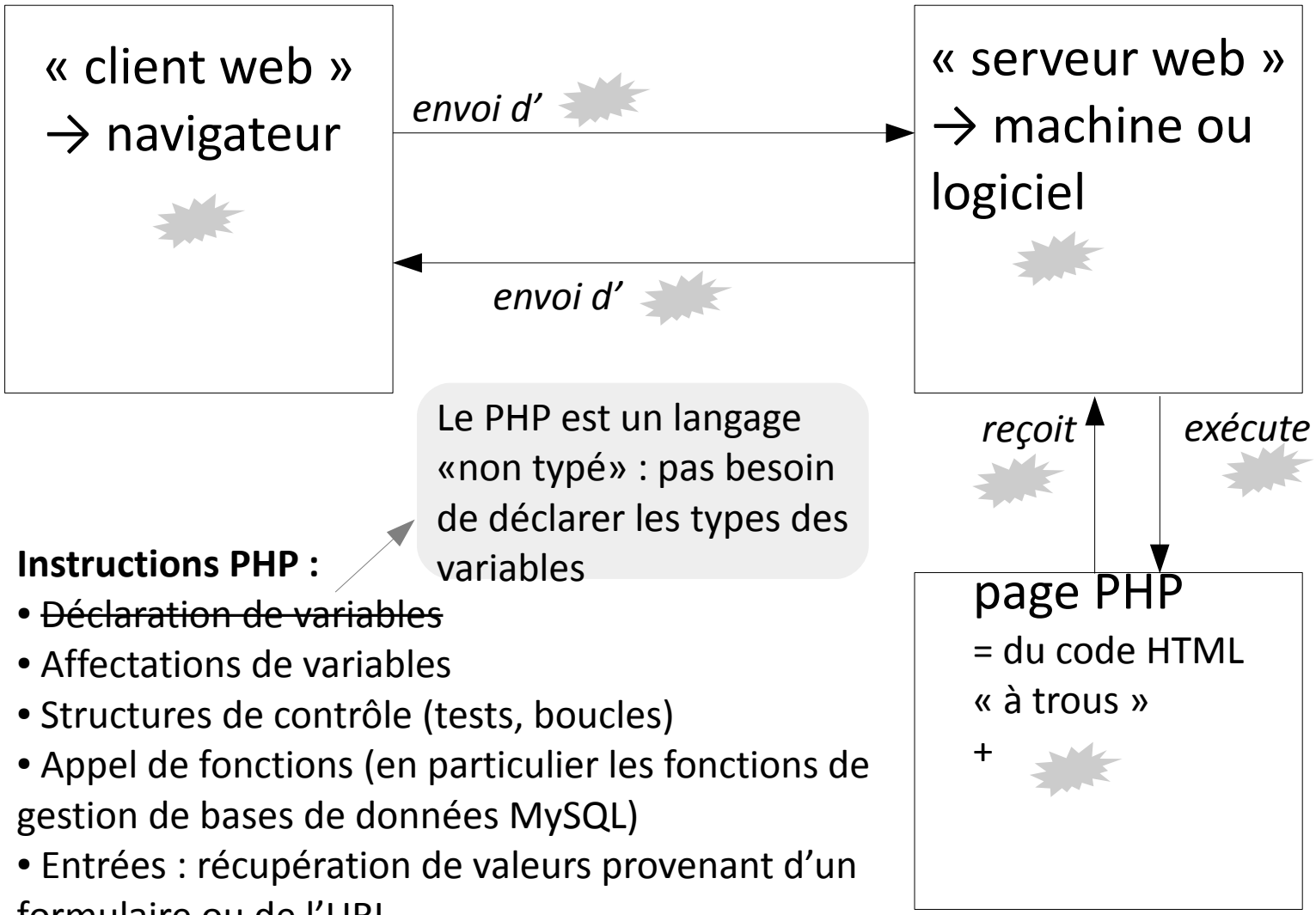
```
public static Color couleurRGB(int r,int g,int b){

  return new Color(r,g,b);

}

public static void dessineRectanglePlein(Graphics g,
int abscisseCoin,int ordonneeCoin,int largeur, int
hauteur, Color couleur) {

  g.setColor(couleur);

  g.fillRect(abscisseCoin, ordonneeCoin, largeur,
  hauteur);

}
```

Syntaxe Java

Utiliser la classe depuis une autre

```
public class BilanSemestre{









}
```

Concept de la programmation web orientée serveur



Le PHP est un langage «non typé» : pas besoin de déclarer les types des variables

Instructions PHP :

- ~~Déclaration de variables~~
- Affectations de variables
- Structures de contrôle (tests, boucles)
- Appel de fonctions (en particulier les fonctions de gestion de bases de données MySQL)
- Entrées : récupération de valeurs provenant d'un formulaire ou de l'URL
- Sorties : affichage (dans les trous du code HTML)

Syntaxe PHP – correspondance avec Java

	Java	PHP
Programme	fichier **`TP1.java`** avec une classe `TP1`, ses propriétés, et ses méthodes dont la méthode **`main`**, qui contiennent des instructions Java	fichier **`TP1.php`** contenant du code HTML (`header, body`) et des balises et contenant des instructions PHP
Compilation	terminal : `javac TP1.java`	pas de compilation : langage « interprété » à la volée
Exécution	terminal : `java TP1 val1 val2`	URL de page PHP dans le navigateur web : `http://mmi.fr/TP1.php ?param1=val1&p2=val2`
Sortie	afficher dans le terminal : `System.out.println( "toto");`	afficher dans le code HTML :

Syntaxe PHP – correspondance avec Java

	Java	PHP

Affichage des paramètres	dans la fonction `main` :
	`public static void main(String[] arg){`
	`System.out.println(`
	`arg[0]+" "+arg[1]);}`
		dans le `body` :
		`<?php`
		`?>`

Initialisation d'une variable	`int i=0;`
	`String x="toto";`
	(inclut la déclaration)
		`$i=0;`
		`$x="toto";` (ou `$x='toto';`)
		(pas de déclaration)

Test	`if(i==0){x="a";}` `else{x="b";}`	

Boucle	`while(i>0){`
	`i++;}`
	
	`for(int i=0;i<10;i++){`
	`…}`
		
		`for($i=0;$i<10;$i++){`
		`…}`

Syntaxe PHP – correspondance avec Java

	Java	PHP

Tableaux	Les numéros de case sont des entiers de 0 à *n*-1 (*n*=nombre de cases)
	La longueur d'un tableau **ne peut pas** changer.
	
	`String[] tab = {"toto","titi"};`
	
	`tab[0]="tototo";`
		Les cases peuvent être désignées par des noms (chaînes de caractères)
		La taille d'un tableau **peut** Changer.
		
		`$tab = array("toto", "titi");`
		
		`$tab[0]="tototo";`

Syntaxe PHP – correspondance avec Java

	Java	PHP

Fonctions	`public static` et le type de sortie précèdent le nom de la fonction, suivi de parenthèses qui contiennent les entrées éventuelles, précédées de leur type.
	
	`public static int somme(int a,int b){`
	`  return a+b;`
	`}`
		Le mot `function` précède le nom de la fonction, suivi de parenthèses qui contiennent les entrées éventuelles.
		
		`function somme($a,$b)`
		`{`
		`  return a+b;`
		`}`

Appel de fonction	Le nom de la fonction est suivi de parenthèses qui contiennent les entrées éventuelles.	Le nom de la fonction est suivi de parenthèses qui contiennent les entrées éventuelles.