ALGORITHMIQUE

Simon Mielcarek

QU'EST CE QUE C'EST ?

- L'**algorithmique** est l'étude et la production de règles et techniques qui sont impliquées dans la définition et la conception d'algorithmes
- Du coup qu'est ce qu'un algorithme ?
  - ≪ Ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations. ≫
- Pour résumer c'est une suite d'instructions, qui une fois exécutées conduit à un résultat donné
- Des exemples ?

QU'EST CE QUE C'EST ?

- Recette de cuisine :
  - Suite d'instructions qui permet d'obtenir un résultat (si exécutées correctement ;))
- Manuel de montage d'un meuble
- Un algorithme peut théoriquement être écrit pour n'importe quel ≪ problème ≫
  - Remplir une casserole
  - Cuire des pâtes
  - Remarquez qu'on peut les combiner
    - Si on sait déjà ≪ remplir une casserole ≫ il devient plus simple de décrire ≪ cuire des pâtes ≫
  - Quelqu'un pour nous écrire l'algorithme de ≪ Cuire des pâtes ≫

QU'EST CE QUE C'EST ?

- Une notion important est que :
  - Un algorithme doit uniquement contenir des instructions compréhensibles par celui qui devra l'éxécuter
  - Recette de cuisine : OK
  - Manuel : OK
  - Plus compliqué quand il s'agit d'écrire un programme ≪ exécutable ≫ par l'ordinateur ?

- Un ordinateur :
  - Les ordinateurs, peu importe la puissance, sont tous aussi ≪ idiots ≫ les uns que les autres
  - Ils ne comprennent que des opérations très simple :
    - Additions, soustractions, affichage, lire, etc

ALGORITHMES ET ORDINATEURS

- On remarque que les algorithmes sont partout
- Pourquoi parle-t-on d'algorithme principalement en informatique ?
  - Un ordinateur est bête comme vu précédemment
  - Mais très fort pour les tâches répétitives
    - Rapide et efficace !
    - Pas de possibilité d'erreur
- En informatique on va principalement parler de traitement de l'information
  - Recherche, comparaison, classement, additions, etc
  - Traiter les informations qui nous entourent

POURQUOI LES ALGORITHMES ?

- Pourquoi utiliser des algorithme en programmation ?
  - Parce qu'ils permettent de résoudre un problème, indépendamment du langage utilisé
  - Cela permet de manipuler la structure logique d'un programme
  - A partir du moment ou l'on ≪ programme ≫ on va utiliser un langage et donc se confronter à une syntaxe et des particularités liées à ce langage
  - En algorithmique on utilise ce que l'on appelle du ≪ pseudo-code ≫
    - C'est une manière de simplifier les instructions afin qu'un humain puisse les comprendre et les retranscrire simplement
    - Exemple :
      ≪ LIRE CLAVIER ≫
      ≪ STOCKER DANS A ≫
      ≪ LIRE CLAVIER ≫
      ≪ STOCKER DANS B ≫
      ≪ AFFICHER A + B ≫
    - Cet algorithme permet tout simplement d'entrer au clavier un nombre a et b et d'afficher le resultat de leur addition

CITATION

≪ Un langage de programmation est censé être une façon conventionnelle de donner des ordres à un ordinateur. Il n'est pas censé être obscur, bizarre et plein de pièges subtils (ça ce sont les attributs de la magie). ≫

Dave Small

DE QUOI SONT CONSTITUÉS LES ALGORITHMES ?

STOCKER L'INFORMATION

- Nous allons voir comment traiter l'information un peu plus tard
- Dans un premier temps parlons un peu du stockage
  - Notion très importante
  - La manière d'organiser les informations et primordiale lors de l'élaboration d'algorithmes

  - Exemple : Dictionnaire
    - Si les mots sont disposés aléatoirement dans le dictionnaire il devient inutilisable

  - Les algorithmes sont composés de structures et de variables

LES STRUCTURES

- Les algorithmes sont principalement constitués 3 types d' éléments
  - Structures d'enchainements
  - D'actions
  - De données

  - Les structure d'enchainements eux aussi se décomposent en 3 sous familles
    - Séquentiel
    - Sélectif
    - Répétitif

LES STRUCTURES

- Séquentielle : Les actions sont effectuées les unes après les autres :
  - Commander une pizza
  - Attendre
  - Manger pizza

- Sélective : Le block est éxecuté si une condition est remplie :
  - SI
  - SINON SI
  - SINON

  - Si ≪ j'ai faim ≫ : je commande une pizza, Sinon si ≪ j'ai soif ≫ : je bois, sinon : rien

LES STRUCTURES

- Répétitive : autrement dit ≪ boucle ≫
  - Execute une action tant qu'une condition est remplie
  - Il y a deux types de boucles :
    - REPETER action JUSQU'À condition
      - Au minimum une itération
    - TANT QUE condition remplie FAIRE action
      - 0 à n itérations

LES DONNÉES

- Une donnée est une information qui sera nécessaire au fonctionnement de l'algorithme
  - Il est défini par 3 caractéristiques
    - Son Type : Entier, Flotant, Booléen, Chaine de caractère
    - Sa valeur
    - Son identifiant : ce qui va permettre de récupérer sa valeur ou de l'utiliser
  - Exemple :
  - NuméroTéléphone nPizzeria = 0320590101

PSEUDO-CODE

PSEUDO-CODE

- C'est une manière de représenter de manière écrite les éléments d'un algorithme
- Chaque élément du code est défini par un terme représentatif et compréhensible par l'homme
- Cela permet de décrire un algorithme sans être lié à un langage

- Ce n'est pas un langage de programmation !!
  - Le langage de programmation lui retranscrit l'algorithme en instruction exécutables par l'ordinateur

LES « BASES » DU PSEUDO CODE

- CONSTANTES
- VARIABLES
  - nombre Entier
  - nbCoups Entier
  - Jouer Char
- Affecter une valeur :
  - A ⟵ B
  - A ⟵ 12
- Lire une valeur (au clavier par exemple)
  - LIRE A
- Afficher une valeur
  - Afficher A

Petits exercices

Exercices partie 1 et 2

Les conditions

Permet de tester si quelque chose est vrai

== égalitée

<> différent

< inférieur à

> supérieur à

! NON

ET

OU

Tables de vérités

Algèbre de boole

ET

Conjonction logique

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Tables de vérités

OU

Disjonction logique

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Tables de vérités

Equivalence

≡

Équivalence logique

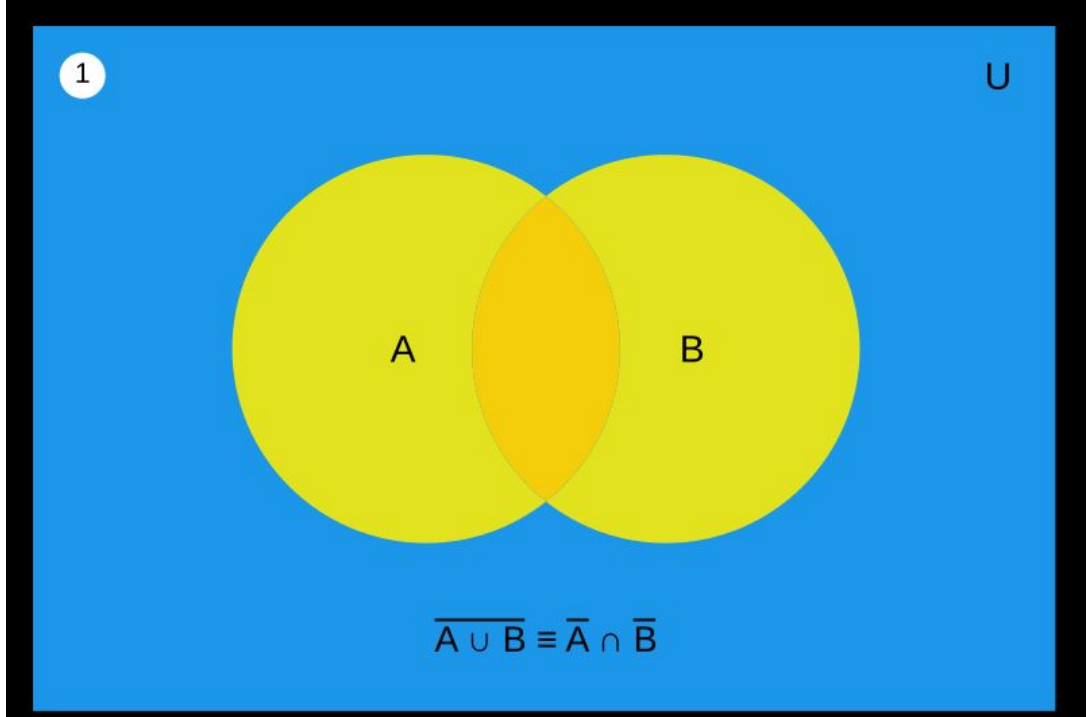
p	q	$p \equiv q$
0	0	1
0	1	0
1	0	0
1	1	1

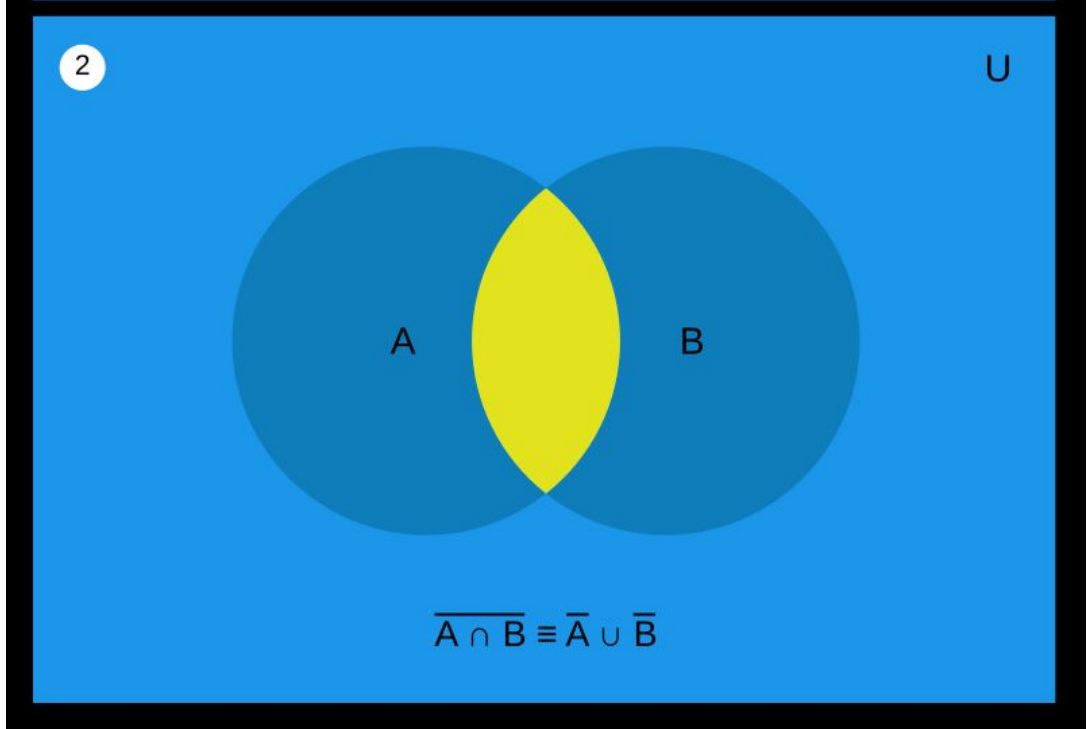
Loi de Morgan

- la négation de la conjonction de deux propositions est équivalente à la disjonction des négations des deux propositions

  ou

- ≪ non(A et B) ≫ est identique à ≪ (non A) ou (non B) ≫


1
U
A
B
$\overline{A \cup B} \equiv \overline{A} \cap \overline{B}$


2
U
A
B
$\overline{A \cap B} \equiv \overline{A} \cup \overline{B}$

LES «BASES » DU PSEUDO-CODE
Boucles

- SI *condition*
  |
  | **ACTION**
  |
  SINON SI *condition*
  |
  | **ACTION**
  |
  SINON

Petits exercices

Exercices partie 3 & 4

LES «BASES » DU PSEUDO-CODE
Boucles

TANT QUE condition
|…

FINTANTQUE

FAIRE
|….

TANT QUE condition

LES «BASES » DU PSEUDO-CODE
Boucles

POUR(Etat Initial, condition, itération)

| …

FIN POUR

Exemple :

POUR(I = 0; i<10; I = I + 1)

…

Petits exercices

Exercice partie 5

PETITS EXERCICES SIMPLES

- Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 10 jusqu'à ce que la réponse convienne.
- Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre.
  - Ex : 3 x 1 = 3 etc…
- Ecrire un algorithme qui demande un nombre puis écrit sa factorielle ainsi que son résultat

PLUS D'EXERCICES

- http://pise.info/algo/enonces1.htm
- http://pise.info/algo/enonces2.htm
- http://pise.info/algo/enonces[...].htm
- http://pise.info/algo/enonces12.htm

- Pour aller plus loin :
  - http://pise.info/algo/codage.htm

LA MODULARITE

- Ecriture d'un programme : création de modules réutilisables
- Exemple concret de module qui sont présent dans quasi tous les langages :
  - Les fonctions mathématiques
  - Quasi tous les langages possèdent une librairie mathématique
  - Ex: fonction Abs : nombre absolu
  - Pow : puissance

- Lorsque nous rédigeons des modules nous partons du fait que nos instructions deviennent basique :
  - Exemple : si j'ai rédigé une fonction pour faire A ^ B s'appelant PUISSANCE
  - Je peux me permettre d'écrire dans mon pseudo code C = PUISSANCE (A, B)

LA MODULARITE

Pourquoi ?

- Facilité d'entretien du code

- Ne pas réinventer la roue

- Tout ce qui est répété devrait avoir sa propre fonction

EXERCICE

- Ecrire le module puissance

- Ecrire un algorithme qui demande à l'utilisateur 2 chiffres A et B puis qui met utilise le module puissance tel que A ^ B
- Demander à l'utilisateur s'il veut faire un autre calcul ou non

DONNÉES ET STRUCTURES DE DONNÉES

DONNÉES

- Comme dit précédemment
  - Types de données primitifs
    - Entier
    - Booléen
    - Chaine
  - Types de données composites
    - Struct
    - Tableau
    - Objet

STRUCT

- Ensemble de données regroupées en une ≪ structure ≫
  - ATTENTION : ce n'est pas un objet !!
  - Ce n'est pas non plus un tableau !!
- Struct : ensemble de données de nature différente représentant une structure
- Pourquoi pas objet ?
  - Pas de méthode !

TABLEAU (ARRAY)

- Ensemble d'éléments du même type
- Ordonné
  - Accessible par son ≪ index ≫

  - Nombre d'éléments finis
  - Contigus
  - Accès rapide : les cellules sont mises en mémoire les une après les autre

  - Avantages : accès immédiat, facile d'utilisation
  - Inconvénients : les cellules sont contiguës donc l'insertion ou la suppression est impossible

TABLEAU : EXEMPLE

On a un tableau de 10 objets

Accéder au premier élément : tableau[0]

Accéder au dernier élément : tableau[9]

Faire un petit algorithme pour afficher les éléments un par un via une boucle.

OBJET

- Conteneur ≪ autonome ≫
- Constitué :
  - D'informations (attributs)
  - De mécanismes (méthodes)

- Il est créé à partir d'un modèle : la classe (ou le prototype en Javascript)

Les Listes

Utilisable comme un tableau :

C’est une liste d’éléments du même type

Moins rapide d’accès

Permet L’insertion et la suppression d’éléments

Chaque élément contient l’information stockée et l’adresse du prochain élément.

Exercices

Exercices partie 6 et 7

EXERCICE

- Proposer un algorithme qui permet de créer calculer la valeur d'un panier dans un magasin A et un magasin B
- Il est possible de saisir autant de données que nécessaire
- L'algorithme sera capable de donner : le magasin le moins cher dans la globalitée, le prix du panier ≪ optimal ≫ si nous allions dans les deux magasins

- A votre avis est-ce qu'il y a besoin d'un tableau ? D'une structure ?

- Faire le même algorithme mais en limitant la saisie à 10 produits
- Donner le magasin qui possède le plus grand nombre de produits qui sont moins chères

EXERCICE

- Le Palindrôme :
  - Ecrire un algorithme qui est capable de dire si une chaîne de caractère est un Palindrome
    - Exemples :
      - NON est un palindrome (il se lit dans les deux sens)
      - esope reste ici et se repose => palindrome
      - Pouet => pas un palindrome
      - Aa => Pas un palindrome
    - Etant donné les limitations d'algobox et pour simplifier l'exercice nous considérons que A != a
    - L'algorithme ne prendra en compte que les caractères alphanumériques et oubliera le reste (ponctuation, espace, etc)

Portée des variables

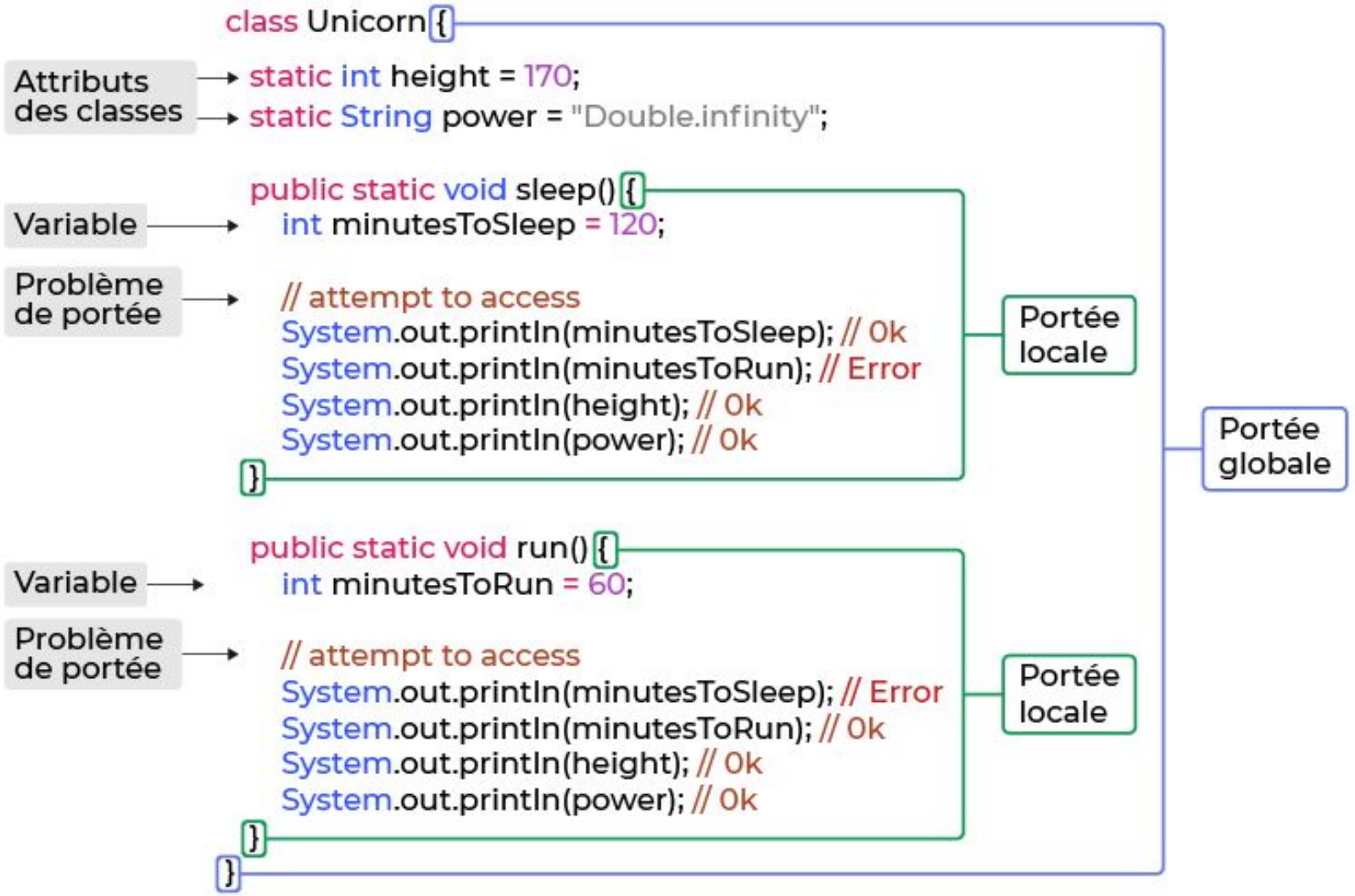
Un bloc est délimité par des accolades

Une variable a une “durée de vie” bornée au bloc dans lequel elle est déclarée

Ce bloc définit la portée de la variable

Une variable est accessible dans les blocs enfants du bloc dans lequel elle est déclarée

L’espace mémoire de la variable est alloué au moment où cette variable est déclarée et libérée à la fin de ce bloc


class Unicorn {
Attributs des classes
static int height = 170;
static String power = "Double.infinity";
public static void sleep() {
Variable
int minutesToSleep = 120;
Problème de portée
// attempt to access
System.out.println(minutesToSleep); // Ok
System.out.println(minutesToRun); // Error
System.out.println(height); // Ok
System.out.println(power); // Ok
}
Portée locale
public static void run() {
Variable
int minutesToRun = 60;
Problème de portée
// attempt to access
System.out.println(minutesToSleep); // Error
System.out.println(minutesToRun); // Ok
System.out.println(height); // Ok
System.out.println(power); // Ok
}
Portée locale
}
Portée globale

ALGOBOX

- Refaire les exercices précédents mais sur AlgoBox
  - Petit outil pour s'entrainer aux algorithmes qui permet de les tester rapidement

S'entraîner en s’amusant

Human Resource Machine

7 billion humans

Exercices supplémentaires

Pour aller plus loin vous pouvez vous entraîner sur les exercices suivants :

http://cours.pise.info/algo/enonces8.htm

http://cours.pise.info/algo/enonces9.htm

http://cours.pise.info/algo/enonces10.htm

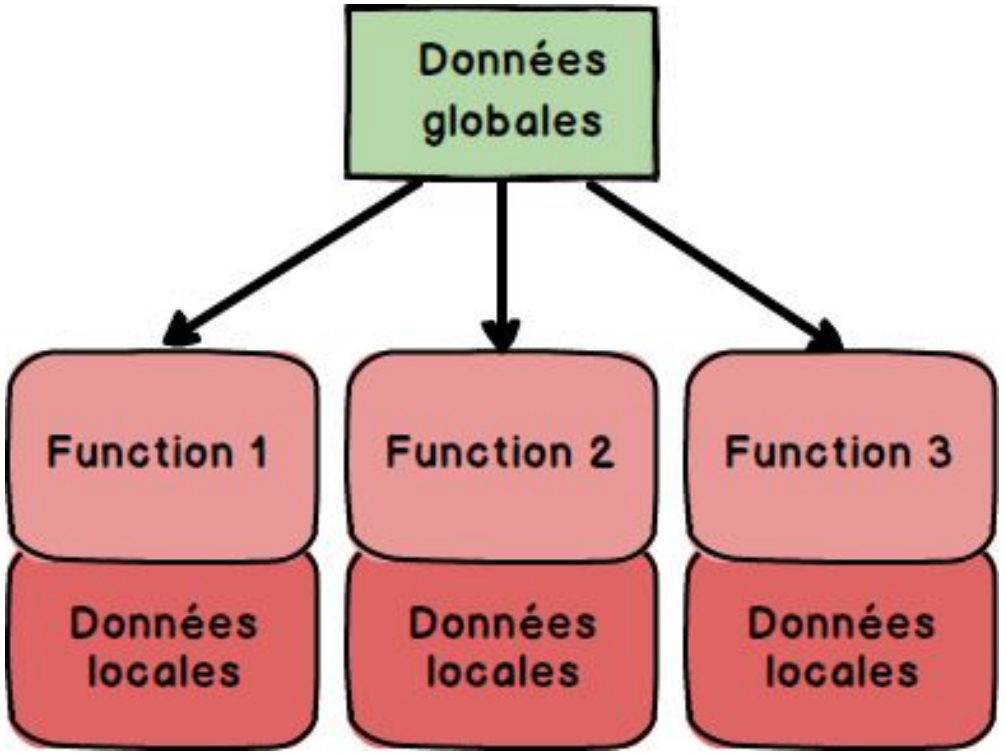
http://cours.pise.info/algo/enonces11.htm

PROGRAMMATION ORIENTÉE OBJET

La Programmation procédurale

Jusque là => Programmation procédurale

Le script s'exécute ligne par ligne et tout est “dans un fichier”



La Programmation procédurale

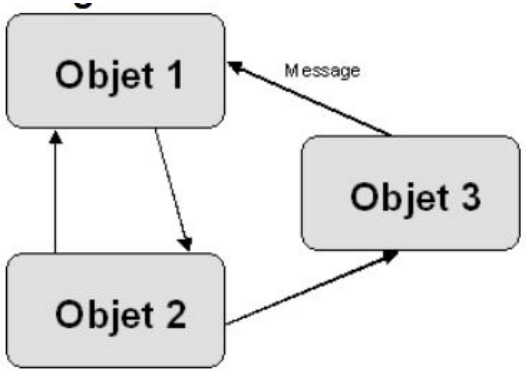
On appelle des procédures qui s'exécutent les unes après les autres

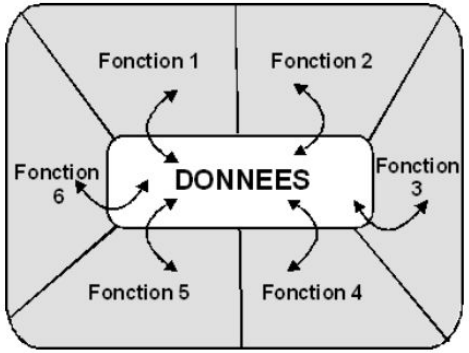
=> Manque de modularité

=> Difficile de traiter correctement les données

Programmation Orientée Objets

Conteneurs indépendants qui interagissent





La Programmation Orientée Objet

Table de comparaison

	Programmation Procédurale	Programmation Orientée Objet
Programmes	Le programme principal est divisé en petites parties selon les fonctions.	Le programme principal est divisé en petit objet en fonction du problème.
Les données	Chaque fonction contient des données différentes.	Les données et les fonctions de chaque objet individuel agissent comme une seule unité.
Permission	Pour ajouter de nouvelles données au programme, l'utilisateur doit s'assurer que la fonction le permet.	Le passage de message garantit l'autorisation d'accéder au membre d'un objet à partir d'un autre objet.
Exemples	Pascal, Fortran	PHP5, C ++, Java.
Accès	Aucun spécificateur d'accès n'est utilisé.	Les spécificateurs d'accès public, private, et protected sont utilisés.
La communication	Les fonctions communiquent avec d'autres fonctions en gardant les règles habituelles.	Un objet communique entre eux via des messages.
Contrôle des données	La plupart des fonctions utilisent des données globales.	Chaque objet contrôle ses propres données.
Importance	Les fonctions ou les algorithmes ont plus d'importance que les données dans le programme.	Les données prennent plus d'importance que les fonctions du programme.
Masquage des données	Il n'y a pas de moyen idéal pour masquer les données.	Le masquage des données est possible, ce qui empêche l'accès illégal de la fonction depuis l'extérieur.

Les Objets

Un **objet** est une entité qui représente (modélise) un élément du domaine étudié

ex : une voiture, un compte bancaire, un nombre complexe, une facture, etc.

Objet = état + actions

Attributs = Etat

Méthodes = Actions

Les Objets

Exemple un compte bancaire :

- Par quoi est-il représenté ?
- Quelles sont ses actions ?

La classe

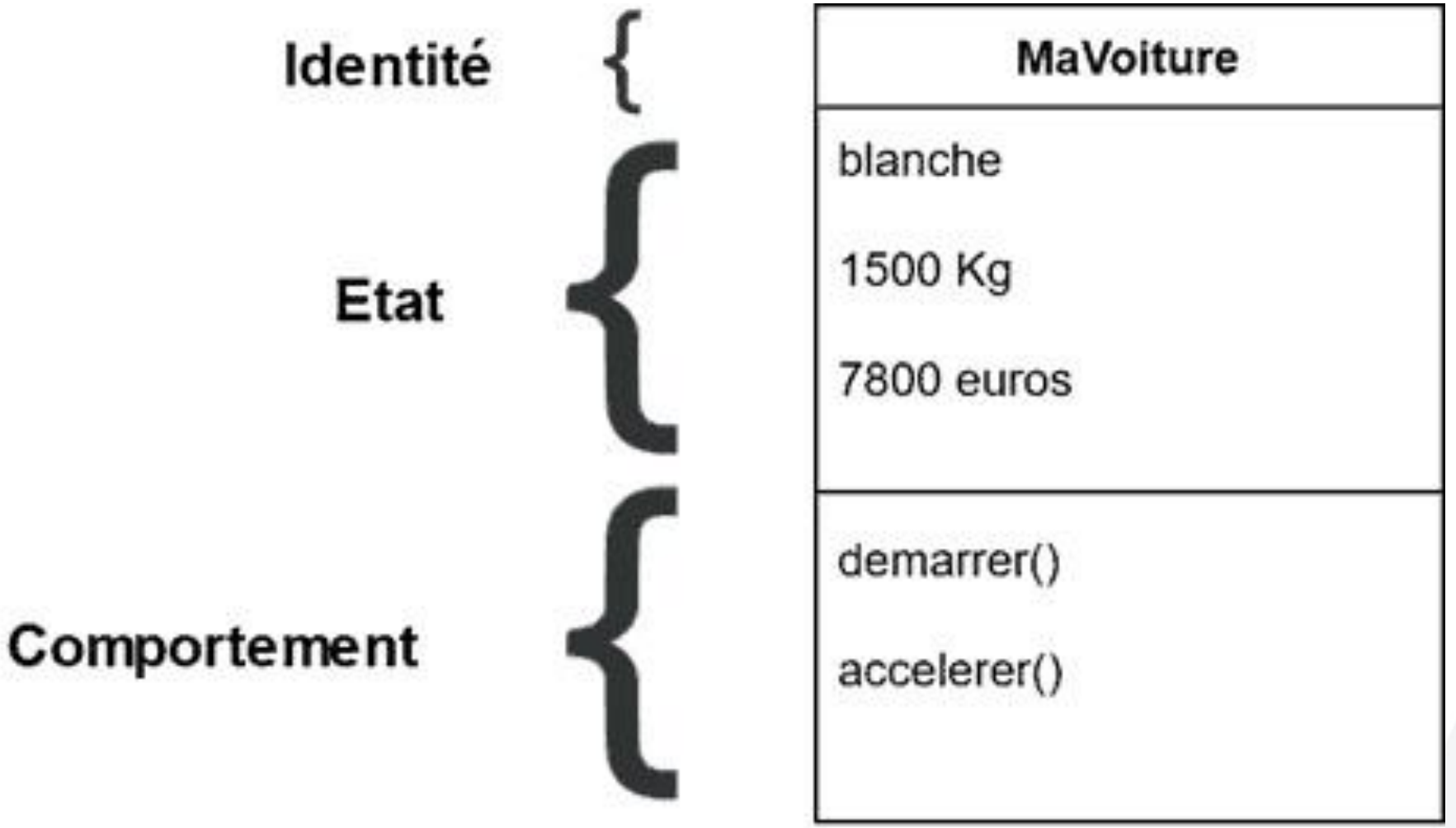
Une classe est un **modèle** d'objet

C’est lui qui est programmé

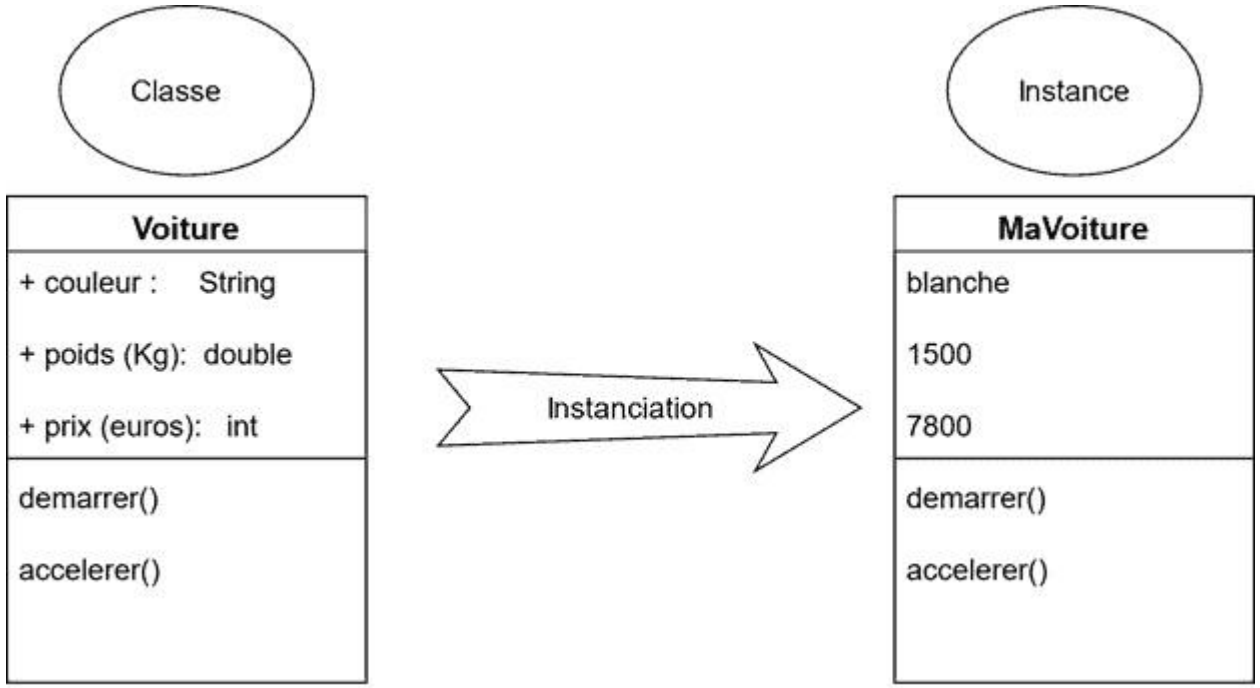
C’est un nouveau Type

Spécifie les **informations** et les **actions** qu'auront en commun tous les objets qui en sont issus.

/!₩ CLASSE ET OBJETS SONT **DIFFÉRENTS**


Identité
Etat
Comportement
MaVoiture
blanche
1500 Kg
7800 euros
demarrer()
accelerer()

Instanciation



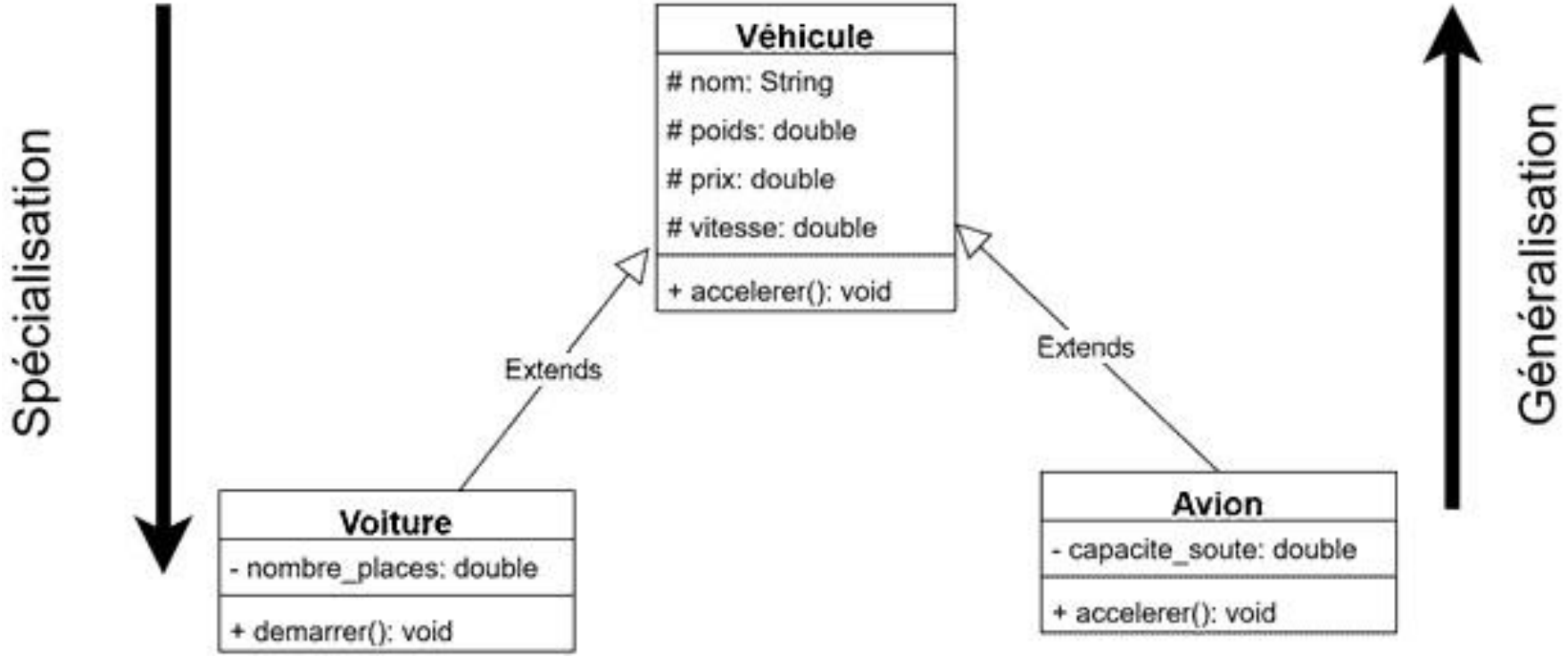
Interactions

Les objets peuvent interagir entre eux

Par exemple :

- CompteBancaire appartient à une Personne
- Voiture est composée de Roues

Héritage


Spécialisation
Véhicule
nom: String
poids: double
prix: double
vitesse: double
+ accelerer(): void
Extends
Extends
Voiture
- nombre_places: double
+ demarrer(): void
Avion
- capacite_soute: double
+ accelerer(): void
Généralisation

ALGO ET TESTS

Algos et Tests Unitaires

Qu’est ce que c’est ?

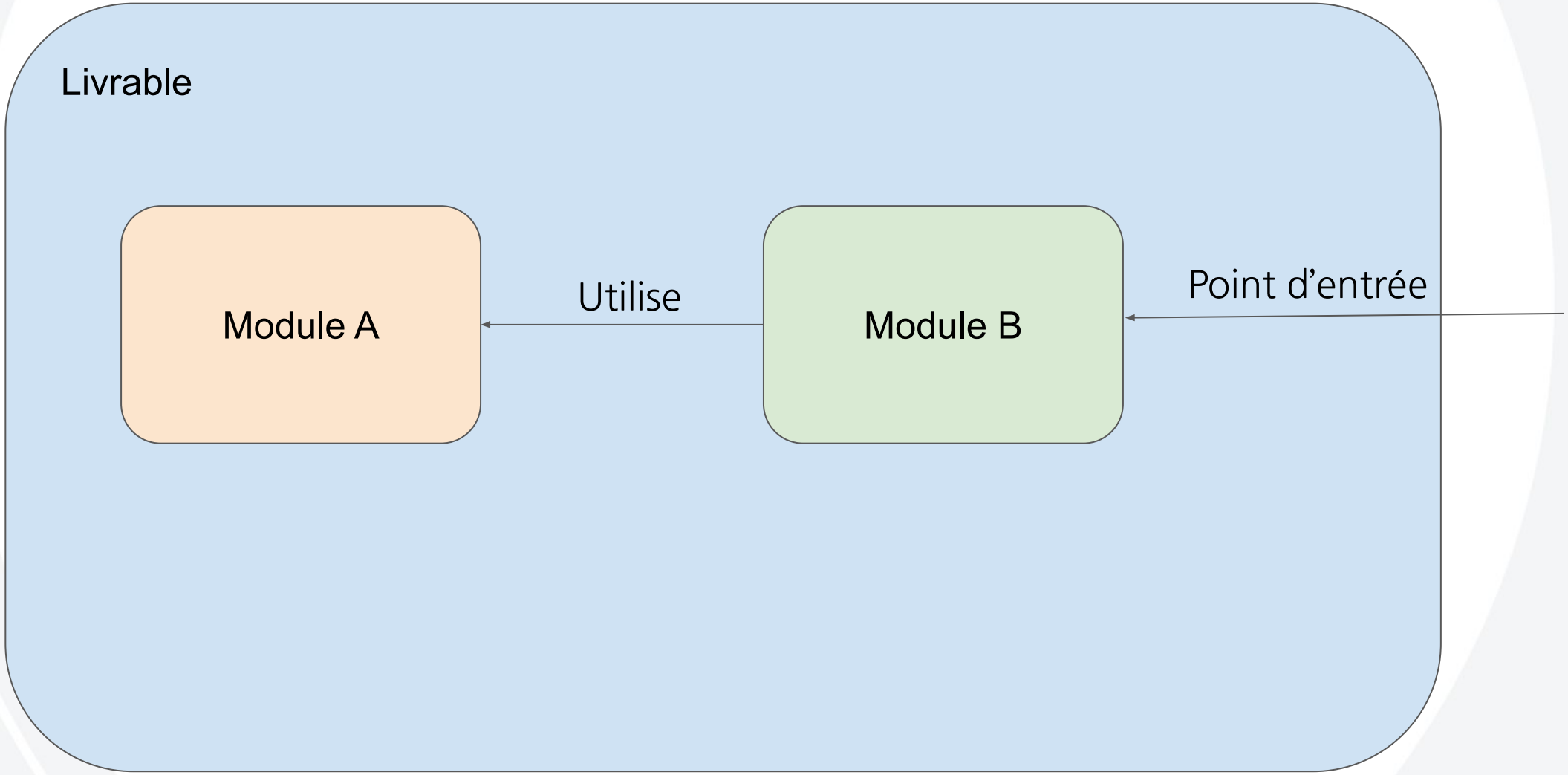
Permet de vérifier le bon fonctionnement de son programme

Les tests en premier, le programme en second !

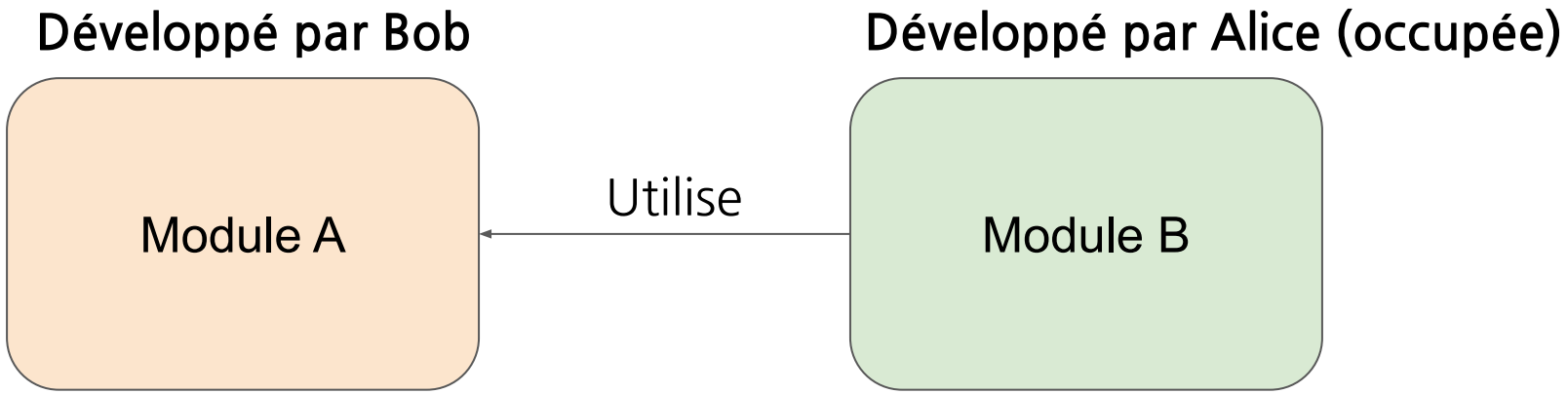
On analyse les situations d'échec et de bon fonctionnement avant de programmer.

1ère étape : coder les situation d’échec

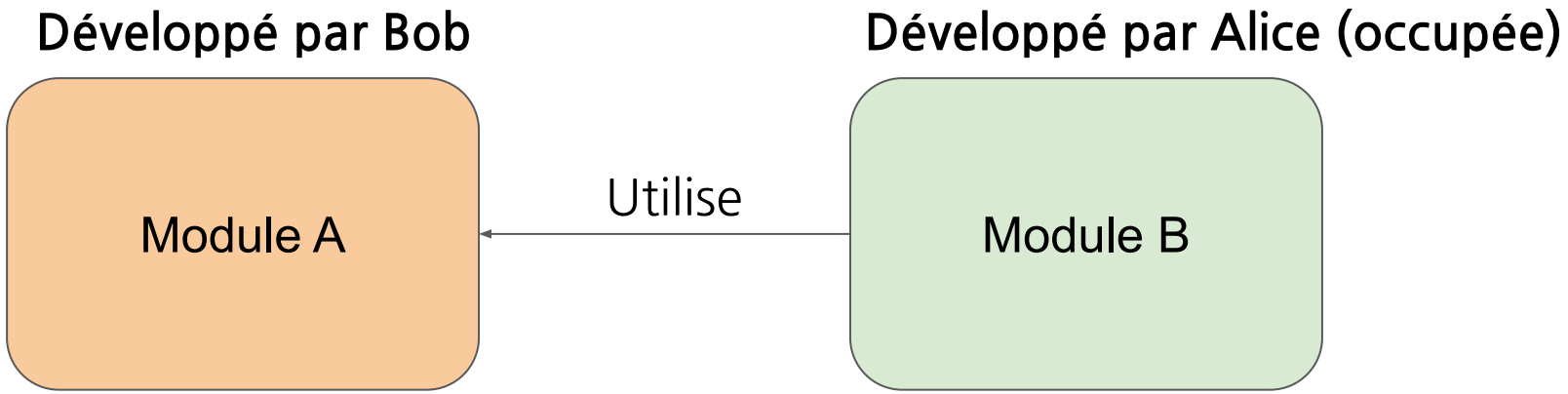
Pourquoi des tests automatisés ?



Pourquoi des tests automatisés ?

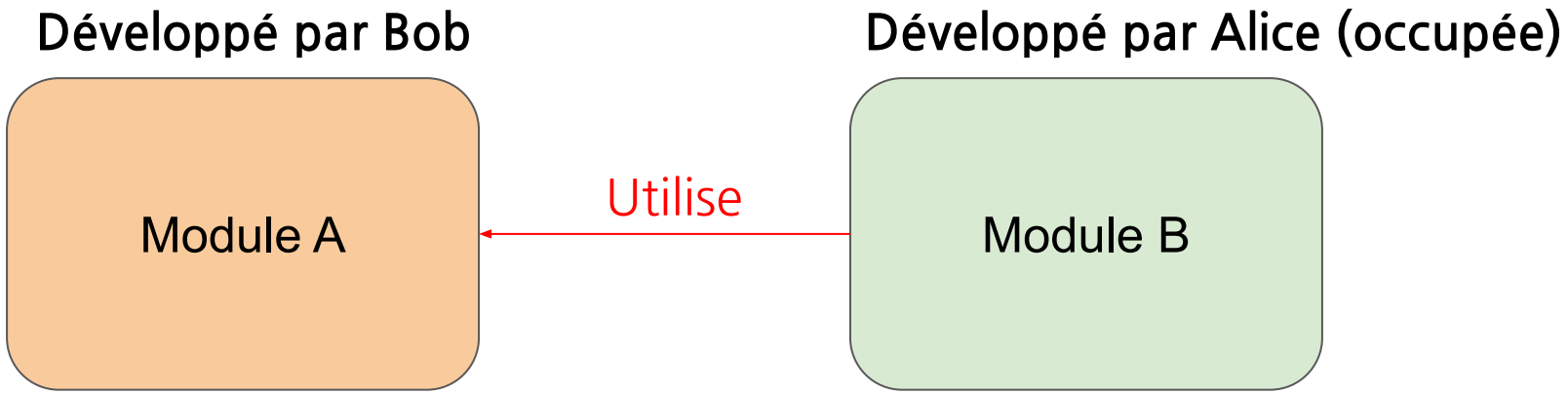


Pourquoi des tests automatisés ?



- Ajout de fonctionnalités
- Résolution des bugs

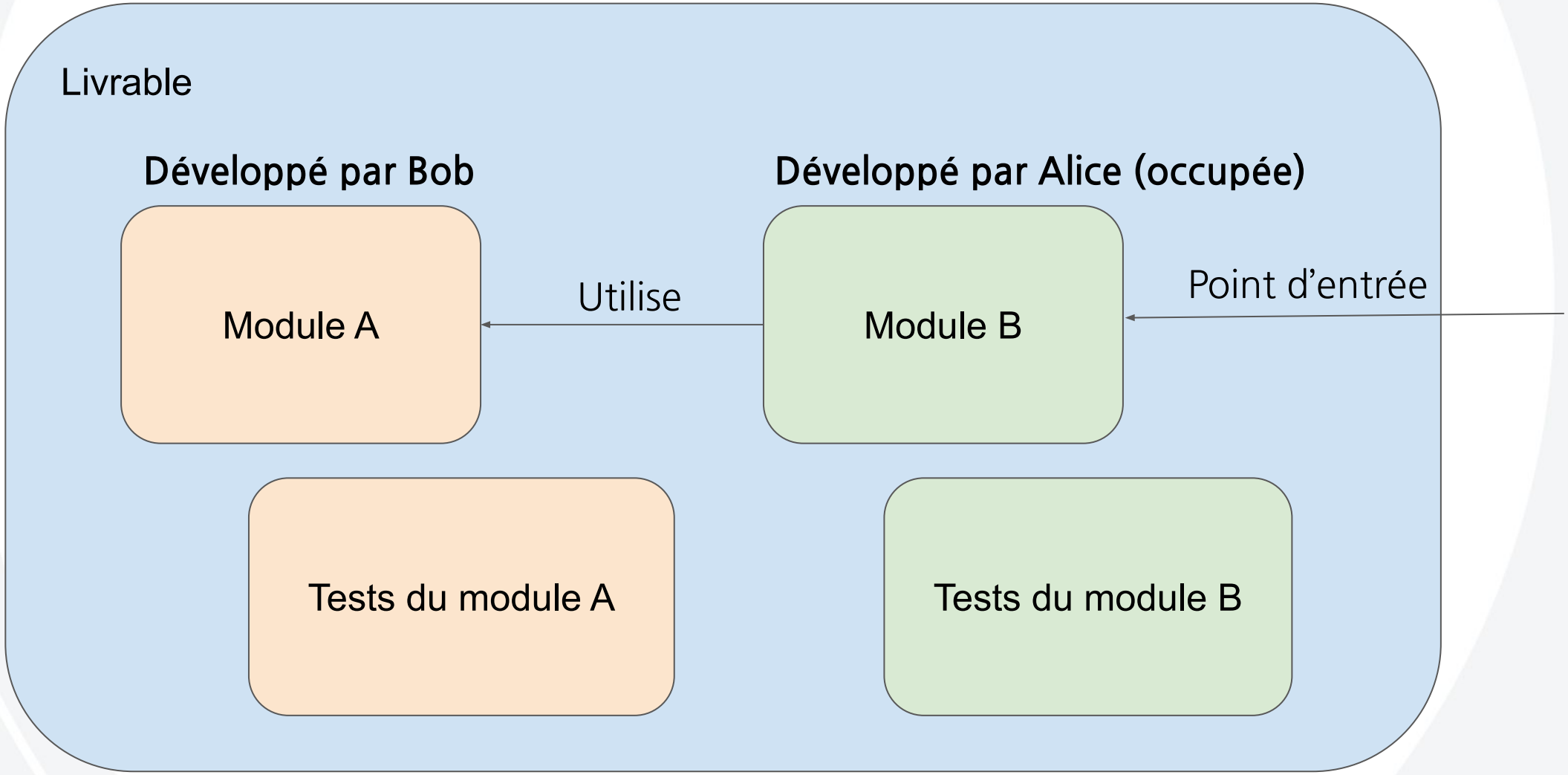
Pourquoi des tests automatisés ?



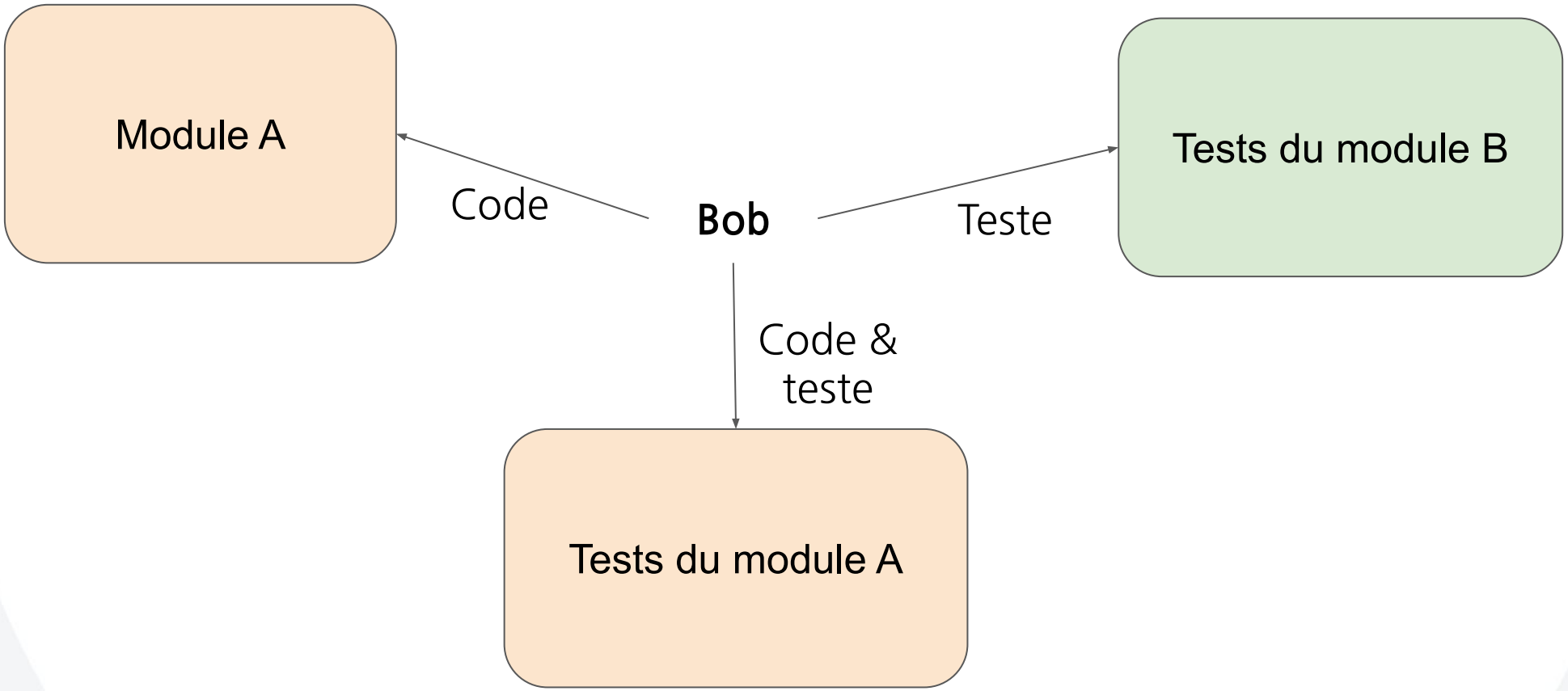
- Ajout de fonctionnalités
- Résolution des bugs

Comment Bob peut-il garantir de ne pas casser le code d'Alice ?

Pourquoi des tests automatisés ?

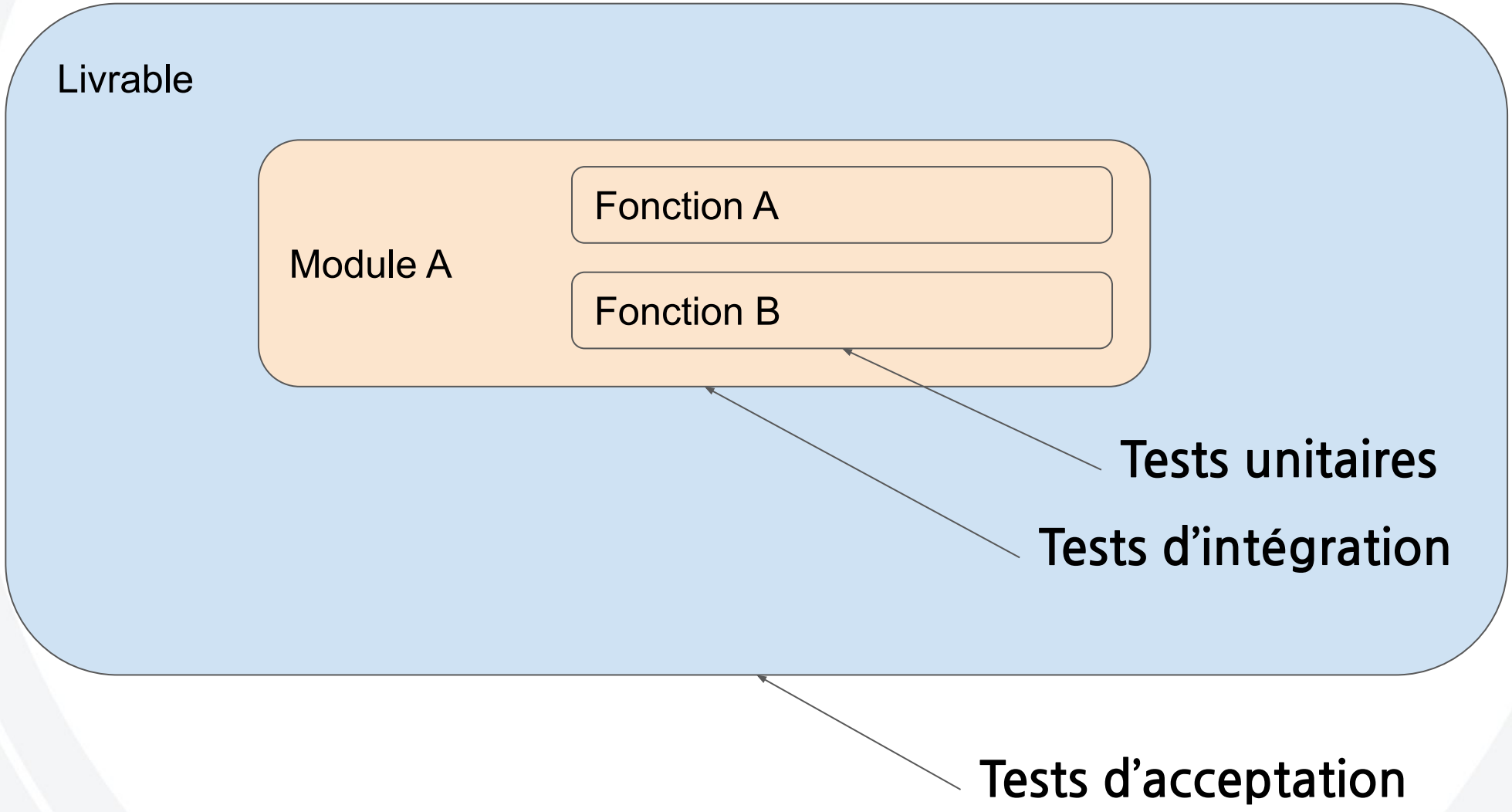


Pourquoi des tests automatisés ?

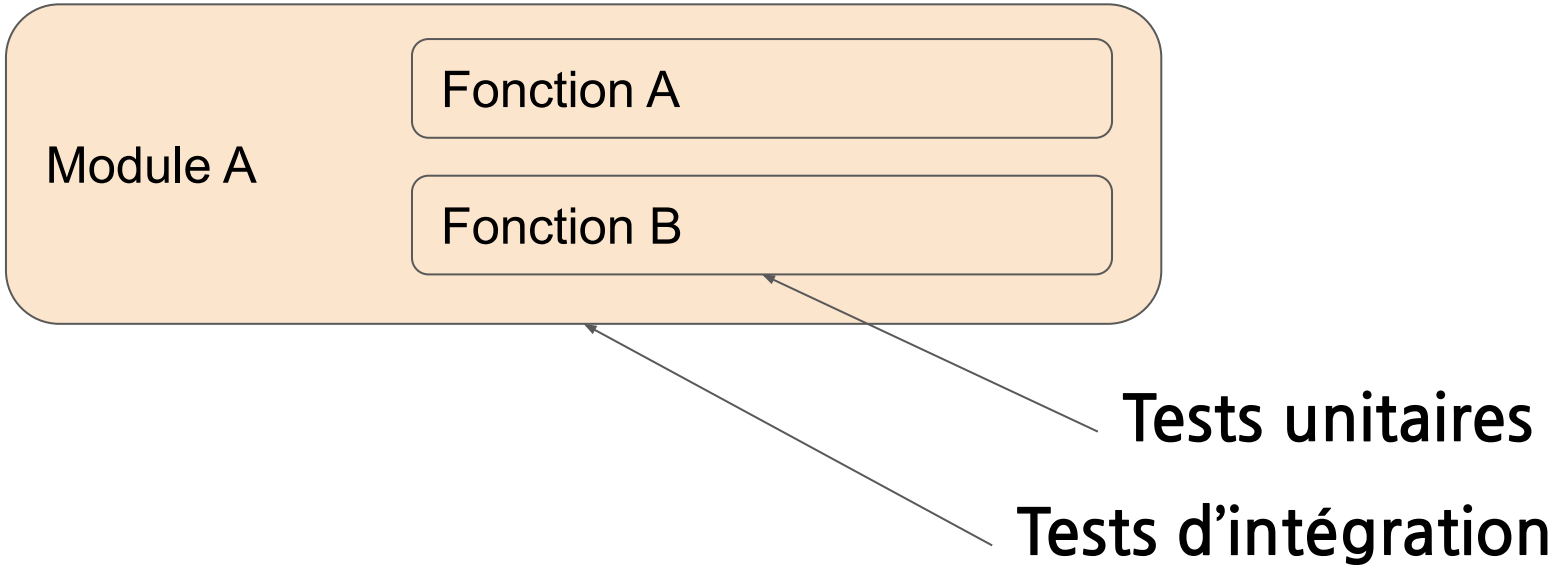


Alice peut garder l'esprit tranquille, Bob gère !

Les différents tests

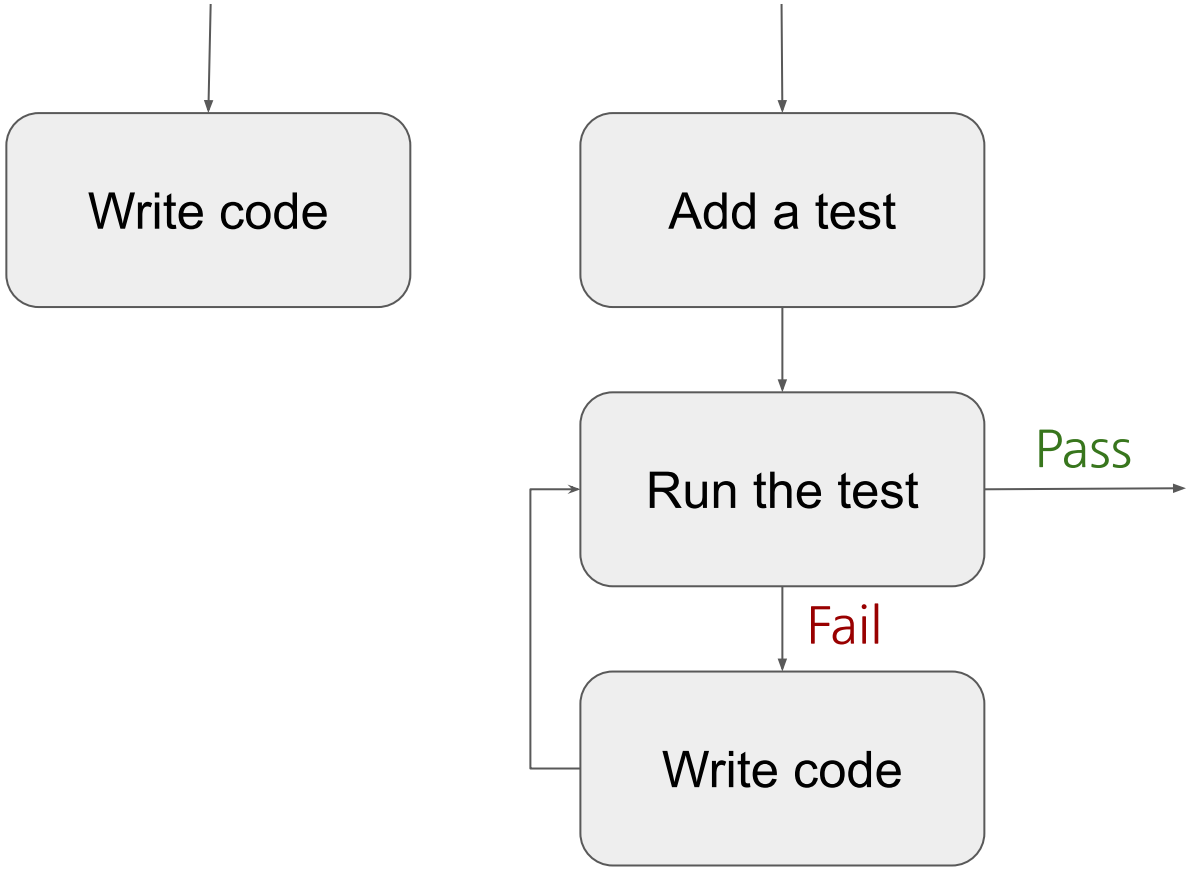


Les tests qui nous intéressent



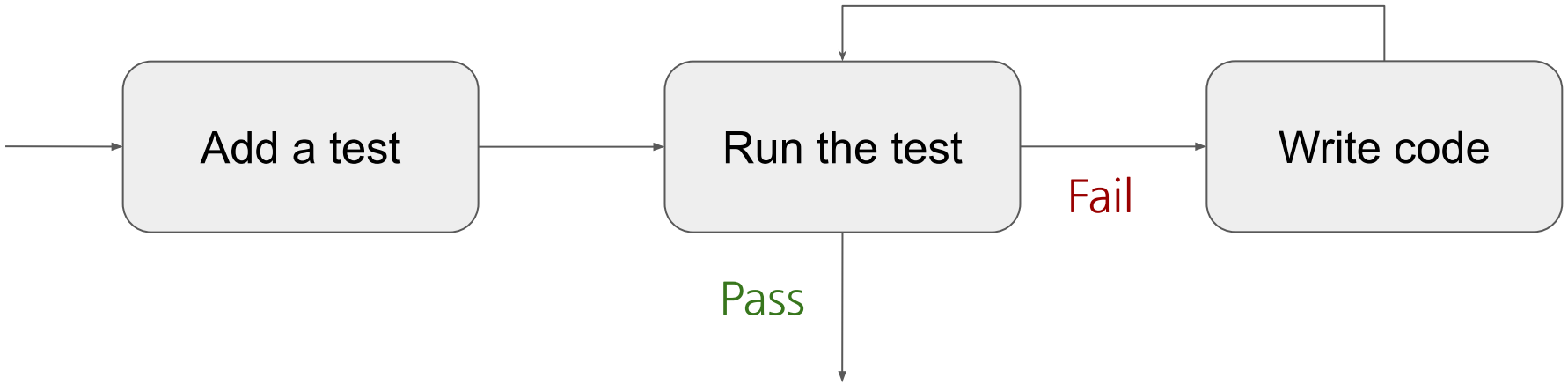
Méthode traditionnelle : on code d'abord

On code, puis plus tard, on écrit un test (peut-être).



“Test First Design”

Consiste à écrire un test avant de coder.



JAVA

- Technologie créé par Sun Microsystems puis racheté par Oracle
- Plusieurs standards (spécifications)
  - Java SE : standard edition
  - Java EE : enterprise edition (extension de SE)
  - Java ME : micro edition

- Java n'est pas qu'un langage !
- Indépendant de la plateforme
  - Machine virtuelle (JVM)

NOTRE PREMIER PROGRAMME !

- Découvrons ensemble l'interface d'Eclipse et créons notre premier programme

```
/** Votre premier programme Java */
class FirstApp {
public static void main (String[] args){
System.out.println("Hello World");
}
}
```

- Pas besoin de plus !