ALGORITHMIQUE · FICHE PRATIQUE

Cheatsheet : Lire les diagrammes algorithmiques

Symboles, chemins, entrées et sorties, décision, fusion et fin d’activité UML.

Par Ali ZAINOUL · Cours gratuit

Un diagramme représente un déroulement avec des formes et des flèches. Il faut annoncer sa notation et ses données avant de suivre un chemin. Cette fiche distingue un organigramme algorithmique d'une activité UML très simple, puis relie chaque schéma à un programme complet.

Une flèche de contrôle indique ce qui peut être exécuté ensuite. Elle n'est pas automatiquement une donnée transmise. Les entrées et les sorties du problème sont précisées séparément des formes de début et de fin.

1. Reconnaître chaque symbole avant le parcours

Dans notre organigramme, un ovale borne le début ou la fin, un losange porte une décision et un rectangle porte un traitement. Une notation d'organigramme peut employer un parallélogramme pour les entrées et sorties ; nous annonçons ici les données dans le contrat et écrivons les affichages comme des traitements.

UML signifie Unified Modeling Language, un langage de modélisation. Une activité est un traitement représenté en étapes, appelées nœuds. Le disque noir marque le départ du flux de contrôle ; le disque entouré d'un cercle termine l'activité représentée. Ces ronds ne contiennent ni la donnée d'entrée ni une valeur de retour.

Légende de l’activité UML utilisée ici

Symbole	Rôle	Erreur à éviter
Disque noir	Nœud initial : départ du contrôle	Le lire comme la quantité zéro

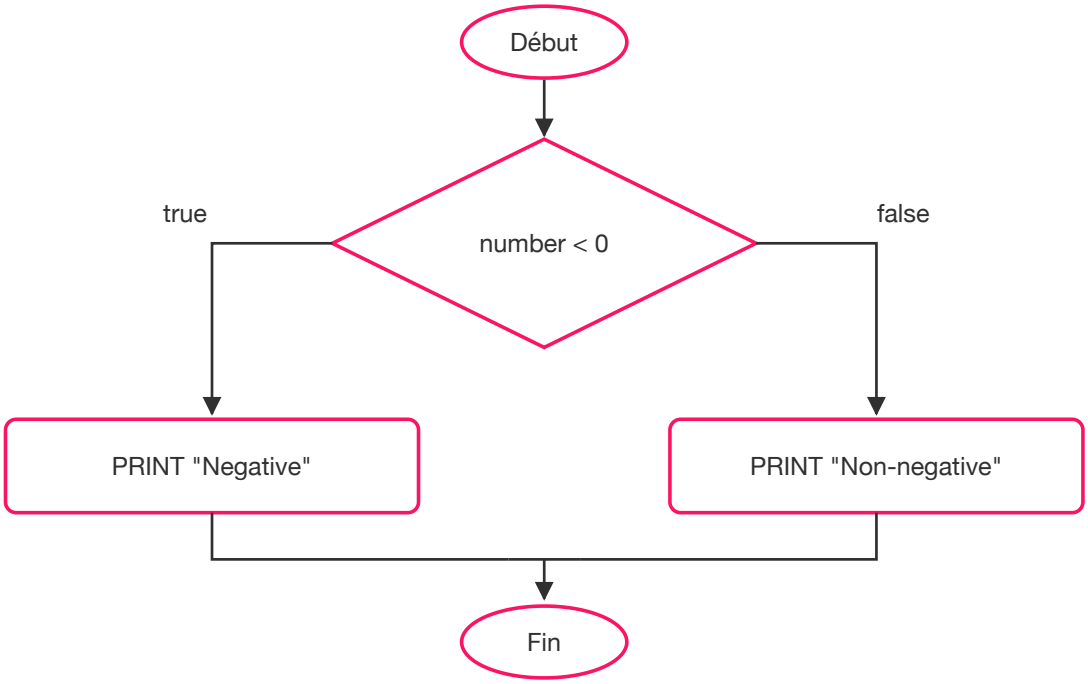
Symbole	Rôle	Erreur à éviter
Rectangle arrondi	Action à exécuter	Confondre le nom de l’action et sa donnée
Losange de décision	Sélection d’un chemin selon une garde	Suivre les deux sorties pour un seul choix
Garde entre crochets	Condition qui autorise une flèche	Déduire la garde de sa seule position gauche/droite
Losange de fusion	Réunion de chemins alternatifs	Attendre deux tâches parallèles
Disque entouré d’un cercle	Fin de l’activité	Le lire comme un port de sortie
Flèche orientée	Sens de poursuite du contrôle	La confondre avec un flux de données

2. Organigramme : classer un nombre sans oublier zéro

L'entrée number est un nombre réel fini, sans infini ni valeur NaN. La sortie est le message Negative si number < 0, et Non-negative sinon. Non-negative signifie non négatif : ce cas comprend zéro. Le message Positive ne serait donc pas un remplacement correct.

Pour −3, le test est vrai : suivez la sortie true, affichez Negative puis rejoignez la fin. Pour zéro, le test est faux et le message est Non-negative. Une seule branche est visitée lors d'un parcours.

Un test de signe : le cas faux comprend zéro.



▼ **Lire les parcours sans animation**

number = -3

1. Début du parcours avec number = -3. Les données sont déjà disponibles.
2. Test 1 : number < 0 donne true (vrai).
3. Action choisie : PRINT "Negative".
4. Les chemins alternatifs se rejoignent. Une seule branche a été exécutée ; il n'y a pas deux tâches à attendre.
5. Fin du parcours. Résultat observable : Negative.

number = 0

1. Début du parcours avec number = 0. Les données sont déjà disponibles.
2. Test 1 : number < 0 donne false (faux).
3. Action choisie : PRINT "Non-negative".
4. Les chemins alternatifs se rejoignent. Une seule branche a été exécutée ; il n'y a pas deux tâches à attendre.

5. Fin du parcours. Résultat observable : Non-negative.

Le même test dans le code

Les trois versions représentent la même décision que le schéma.

Pseudo-code

```
// example.pseudo
// classify_sign
number <- -3
IF number < 0 THEN
    PRINT "Negative"
ELSE
    PRINT "Non-negative"
END IF
```

Python

```python
# classify_sign.py
number = -3
if number < 0:
    message = "Negative"
else:
    message = "Non-negative"
print(message)
```

`if` choisit la branche ; `else` couvre les nombres nuls ou positifs du domaine annoncé.

MATLAB

```
% classify_sign.m
number = -3;
if number < 0
    message = 'Negative';
else
    message = 'Non-negative';
end
disp(message);
```

`end` ferme le choix ; `disp` affiche ensuite le message retenu.

> **Résultat expliqué.** −3 produit Negative. Zéro et 4 produisent Non-negative.

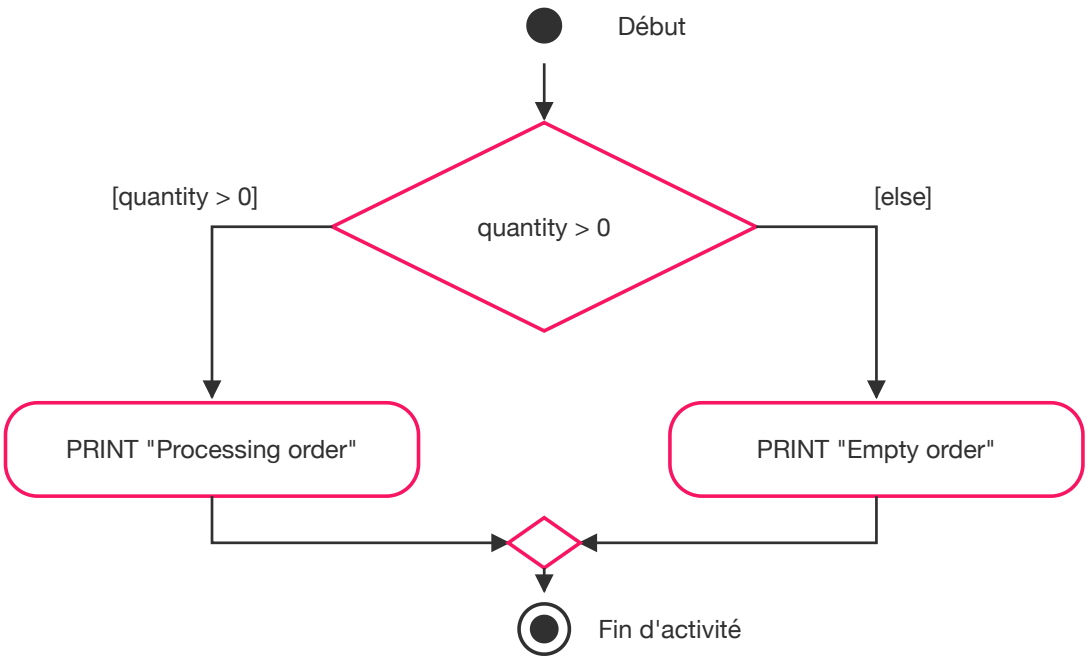
3. UML : lire gardes, action et fusion

L'entrée quantity est un entier non négatif disponible au démarrage. La sortie observable est un affichage : Processing order si la quantité est positive, Empty order si elle vaut zéro. Le programme n'effectue pas un paiement ni un véritable traitement de commande.

Une garde est une condition écrite entre crochets sur une flèche. [quantity > 0] autorise la première branche. [else] autorise l'autre quand aucune autre garde de cette décision n'est vraie. Dans notre domaine, cette branche correspond à zéro.

Le losange supérieur choisit ; le losange inférieur fusionne les alternatives. La fusion n'attend pas l'exécution des deux actions : une seule a eu lieu. L'arrivée au cercle final termine l'activité. Les broches UML, petits carrés d'entrée ou de sortie d'une action, ne sont pas utilisées dans ce premier dessin.

Une décision exclusive suivie d'une fusion et d'une fin d'activité.



▼ **Lire les parcours sans animation**

quantity = 2

1. Début du parcours avec quantity = 2. Les données sont déjà disponibles.
2. Test 1 : quantity > 0 donne true (vrai).
3. Action choisie : PRINT "Processing order".
4. Les chemins alternatifs se rejoignent. Une seule branche a été exécutée ; il n’y a pas deux tâches à attendre.
5. Fin du parcours. Résultat observable : Processing order.

quantity = 0

1. Début du parcours avec quantity = 0. Les données sont déjà disponibles.
2. Test 1 : quantity > 0 donne false (faux).
3. Action choisie : PRINT "Empty order".
4. Les chemins alternatifs se rejoignent. Une seule branche a été exécutée ; il n’y a pas deux tâches à attendre.
5. Fin du parcours. Résultat observable : Empty order.

Observer le choix d’une quantité

L’action du schéma est rendue visible par un message, avec les mêmes deux cas dans les langages.

Pseudo-code

```
// example.pseudo
// process_quantity
quantity <- 2
IF quantity > 0 THEN
    PRINT "Processing order"
ELSE
    PRINT "Empty order"
END IF
```

Python

```python
# process_quantity.py
quantity = 2
if quantity > 0:
    print("Processing order")
else:
    print("Empty order")
```

MATLAB

```
% process_quantity.m
quantity = 2;
if quantity > 0
    disp('Processing order');
else
    disp('Empty order');
end
```

> **Résultat expliqué.** La quantité 2 produit Processing order ; la quantité 0 produit Empty order.

4. Vérifier un diagramme comme une implémentation

Pour relire un schéma, nommez l'entrée, calculez chaque test puis suivez seulement les flèches autorisées. Comparez la sortie à celle du code pour la même entrée. Une jolie disposition ne corrige pas une condition manquante.

Dans un schéma de boucle, le retour doit rejoindre le test après une mise à jour. Revenir vers une initialisation qui rétablit toujours la même valeur peut empêcher l'arrêt. Une animation aide à lire un cas ; elle ne constitue pas une preuve de tous les comportements possibles.

> **Erreur à éviter.** Deux gardes quantity >= 0 et quantity <= 0 se recouvrent pour zéro. Pour ce choix exclusif, elles ne donnent pas les deux alternatives annoncées.

▼ La flèche sortant du disque initial représente-t-elle quantity ? Le cercle final représente-t-il le message affiché ?

Non. Ces éléments représentent le contrôle : démarrer puis terminer le parcours. quantity est une donnée d’entrée ; le texte affiché est une sortie du problème. Il faut les distinguer des flèches et des ronds.

Télécharger cette fiche en PDF · Toutes les fiches par thème

Version web : https://www.clearcode.fr/courses/programming/algorithmic/cheatsheet/representations