Dieu a-t-il programmé le monde en Java ? Monadologie leibnizienne et programmation orientée objet *

Baptiste Mélès

26 août 2015

On s'imagine souvent qu'écrire des programmes pour un ordinateur consiste avant tout à taper, le dos voûté, des suites de 0 et de 1. C'est parler le langage de la machine, qui, comme chacun croit savoir, ne parle que le langage binaire.

Mais la programmation est tout l'inverse. Dans la plupart des cas, c'est la machine qui parle le langage des humains. Non seulement les langages de programmation sont légion, mais il existe surtout de nombreux styles ou « paradigmes » de programmation [1]. Ces styles de programmation sont autant de manières, pour le programmeur, de se représenter et de décrire la tâche que la machine doit effectuer ; ce sont pour lui des façons de penser le monde.

Avant même de taper la moindre ligne de son programme, le programmeur doit ainsi, nez en l'air ou crayon à la main, interroger ses propres représentations et déterminer le cadre dans lequel il convient de les exprimer. Au moment de choisir un style de programmation, le programmeur doit prendre position sur des questions philosophiques aussi ambitieuses que « les concepts existent-ils ? » et « les concepts existent-ils indépendamment des choses ou bien seulement de façon immanente aux choses ? ». Ce niveau d'abstraction relève de la plus pure métaphysique.

Nous montrerons ici que la programmation orientée objet repose sur certaines intuitions métaphysiques qui se trouvent exprimées notamment dans la philosophie « monadologique » de Gottfried Wilhelm Leibniz (1646–1716). Après avoir rappelé les principales thèses de la monadologie leibnizienne, nous montrerons ici en quoi les concepts fondamentaux de la programmation orientée objet permettent d'en donner une expression formelle.

*Une première version de ce texte a été diffusée le 10 mars 2015. L'auteur remercie Raphaël Fournier-S'niehotta pour ses précieuses remarques.

1. Il existe ainsi la programmation impérative, la programmation structurée, la programmation fonctionnelle, la programmation parallèle, la programmation logique, la programmation orientée objet, etc. — celles-ci pouvant souvent être combinées.

1 La monadologie leibnizienne

1.1 Les monades

La doctrine des monades est principalement exposée dans deux textes écrits par Leibniz en 1714 et publiés quelques années après sa mort : les *Principes de la nature et de la grâce* (PNG) et le texte édité sous le nom de *Monadologie* (ML).

Selon Leibniz, la nature est composée de substances qui peuvent être de deux sortes : les substances simples, qu'il appelle monades, et les substances composées, qui sont les réalités visibles à notre échelle.

> 1. La monade, dont nous parlerons ici, n'est autre chose qu'une substance simple, qui entre dans les composés ; *simple*, c'est-à-dire sans parties.
>
> 2. Et il faut qu'il y ait des substances simples, puisqu'il y a des composés ; car le composé n'est autre chose qu'un amas, ou *aggregatum* [agrégat] des simples. (ML 1–2 ; voir aussi PNG 1)

Dans la mesure où les monades sont des éléments qui composent tous les corps et qui ne sont eux-mêmes pas composés, elles peuvent être vues comme « les véritables Atomes de la Nature » (ML 3). Mais contrairement à l'atome des philosophes antiques — Leucippe, Démocrite, Épicure —, qui est inerte, la monade ne cesse de se modifier dans le temps :

> tout être créé est sujet au changement, et par conséquent la Monade créée aussi, et même [...] ce changement est continuel dans chacune (ML 10).

1.2 Perceptions et appétitions

Les modifications de la monade ne résultent pas de l'effet d'une action extérieure, mais de son activité propre : à la différence de l'atome, elle est capable de se modifier par elle-même. Une monade ne se caractérise donc non seulement par ses « qualités », c'est-à-dire ses propriétés — « autrement ce ne seraient pas même des êtres » (ML 8) —, mais aussi par ses « actions internes » :

> une Monade en elle-même, et dans le moment, ne saurait être discernée d'une autre que par les qualités et actions internes, lesquelles ne peuvent être autre chose que ses *perceptions* (c'est-à-dire, les représentations du composé, ou de ce qui est dehors, dans le simple) et ses *appétitions* (c'est-à-dire, ses tendances d'une perception à l'autre) qui sont les principes du changement. (PNG 2)

Les perceptions sont ainsi les représentations que se fait la monade de ce qui lui est extérieur ; les appétitions sont l'activité propre de la monade, source de son changement — typiquement le désir ou la volonté. Si la perception

est l'action du monde sur la monade, l'appétition est l'action de la monade sur le monde.

Plus précisément, on peut distinguer deux types de perception, l'une externe et l'autre interne, cette dernière étant appelée *apperception* ou *conscience* :

> il est bon de faire distinction entre la *Perception* qui est l'état intérieur de la Monade représentant les choses externes, et l'*Apperception* qui est la *Conscience*, ou la connaissance réflexive de cet état intérieur, laquelle n'est point donnée à toutes les âmes, ni toujours à la même âme. (PNG 4 ; voir aussi ML 14–15)

La monade peut ainsi posséder une conscience de soi.

L'ensemble des interactions entre la monade et le monde extérieur, telles qu'elles sont vécues du point de vue de la monade, se résume aux perceptions et appétitions. Aussi peut-on directement en dériver une certaine image du monde :

> Chaque monade est un miroir vivant, ou doué d'action interne, représentatif de l'univers, suivant son point de vue, et aussi réglé que l'univers lui-même (PNG 3).

Mais cette image reste toujours interne à la monade et centrée sur elle. Jamais celle-ci ne sort d'elle-même, c'est-à-dire de ses perceptions et de ses appétitions :

> les Monades n'ont point de fenêtres, par lesquelles quelque chose y puisse entrer ou sortir. [...] Ainsi ni substance ni accident ne peut entrer de dehors dans une monade. (ML 7)

Tout changement vient donc nécessairement de l'intérieur :

> les changements naturels des Monades viennent d'un *principe interne*, puisqu'une cause externe ne saurait influer dans son intérieur (ML 11).

Une monade ne peut agir directement sur une autre ; tout au plus peut-elle avoir l'appétition de changer l'état d'autres monades, mais ce sont elles qui se modifieront de façon strictement interne. Tout se passe comme si l'une envoyait un message que l'autre recevait au moyen d'une transmission adéquate, et qui lui intime d'effectuer telle ou telle action interne. Lorsqu'on dit qu'une monade agit sur l'autre,

> ce n'est qu'une influence *idéale* d'une Monade sur l'autre, qui ne peut avoir son effet que par l'intervention de Dieu, en tant que dans les idées de Dieu une Monade demande avec raison que Dieu en réglant les autres dès le commencement des choses, ait égard à elle. Car puisqu'une Monade créée ne saurait avoir une influence physique sur l'intérieur de l'autre, ce n'est que par ce moyen que l'une peut avoir de la dépendance de l'autre. (ML 51)

Ainsi les monades n'agissent-elles les unes sur les autres que par l'entremise de Dieu.

1.3 L'harmonie préétablie

Si une monade ne peut interagir directement avec les autres monades, comment deux monades différentes peuvent-elles avoir connaissance de leurs actions respectives ? Comment les messages sont-ils transmis ? La cohérence entre leur émission et leur réception doit être garantie. Ainsi Leibniz formule-t-il la théorie de l'harmonie préétablie :

> il y a une *harmonie* parfaite entre les perceptions de la Monade et les mouvements des corps, préétablie d'abord entre le système des causes efficientes [c'est-à-dire la causalité à l'œuvre dans la nature] et celui des causes finales [c'est-à-dire les buts poursuivis par les êtres vivants].

Les monades n'agissent pas directement les unes sur les autres, mais sont synchronisées à l'avance comme des horloges. C'est seulement par ce réglage préalable que

> entre les créatures les actions et passions [2] sont mutuelles. Car Dieu, comparant deux substances simples, trouve en chacune des raisons qui l'obligent à y accommoder l'autre, et par conséquent ce qui est actif à certains égard est passif suivant un autre point de considération (ML 52).

L'action d'une monade est ainsi la passion d'une autre.

N'étant pas composées de substances plus petites, les monades ne peuvent être formées par agrégat, mais seulement créées en passant *ex abrupto* du néant à l'être :

> les Monades ne sauraient commencer ni finir que tout d'un coup, c'est-à-dire elles ne sauraient commencer que par création et finir que par annihilation, au lieu que ce qui est composé commence ou finit par parties. (ML 6)

La cause ultime de leur existence est appelée Dieu. Possédant toutes les perfections, Dieu a notamment « la puissance, la connaissance, et la volonté parfaites » (PNG 9) ; il n'a donc pu construire l'univers que selon « le meilleur plan possible », posant le cadre spatio-temporel et les lois du mouvement les plus adéquates :

> Il suit de la perfection suprême de Dieu qu'en produisant l'univers il a choisi le meilleur plan possible, où il y ait la plus grande variété, avec le plus grand ordre : le terrain, le lieu, le temps, les mieux ménagés ; le plus d'effet produit par les voies les plus simples ; le plus de puissance, le plus de connaissance, le plus de bonheur et de bonté dans les créatures que l'univers en pouvait admettre. (PNG 10)

2. Le terme de « passions » désigne ici les états *passifs* de la monade, c'est-à-dire l'ensemble de ce qu'elle subit, par opposition aux actions. Les « passions » au sens moderne, comme la jalousie et la peur, n'en sont qu'un cas particulier.

Puisque le monde est le meilleur possible — ce qui ne veut pas dire qu'il soit parfait, car le monde parfait pouvait hélas être impossible — et que chaque monade est une image du monde, la perfection du tout résonne en chaque individu :

> chaque Monade, chaque centre substantiel, doit avoir ses perceptions et ses appétits les mieux réglés qu'il est compatible avec tout le reste. (PNG 12)

Chaque monade est réglée pour s'harmoniser le plus parfaitement possible avec le reste du monde.

1.4 Les essences

Parmi ses attributions, Dieu ne fait pas seulement passer les monades du néant à l'être, mais c'est également lui qui fixe à l'avance leur nature :

> En Dieu est non seulement la source des existences, mais encore celle des essences [...]. C'est parce que l'Entendement de Dieu est la région des vérités éternelles, ou des idées dont elles dépendent (ML 43).

Dans l'entendement de Dieu, les essences sont hiérarchisées par genres et par espèces. Par exemple, certains « corps » sont des vivants appelés « animaux », qui se divisent en « bêtes » — les animaux incapables de raisonnement — et en « esprits » — ceux qui en sont doués (PNG 4–5). Les essences sont ordonnées selon une structure arborescente.

Plus encore : Dieu ne détermine pas seulement l'*essence générale*, c'est-à-dire le concept dont relève l'individu (par exemple le concept de chien), mais également l'*essence individuelle*, c'est-à-dire la totalité des propriétés qui caractérisent un individu (par exemple Médor). Créées par Dieu de toute éternité, les deux sortes d'essences ont une réalité en tant qu'elles préexistent, dans son entendement, aux individus.

Pour résumer, la théorie leibnizienne des monades est une doctrine métaphysique dans laquelle les concepts, créés par Dieu, sont organisés de façon hiérarchique et préexistent aux individus ; les individus sont des êtres atomiques et animés, qui, quoique enfermés sur eux-mêmes, n'en sont pas moins capables de se représenter le monde et d'agir sur lui en échangeant des messages grâce à l'action intermédiaire d'une harmonie préétablie par le créateur.

Nous allons voir que cette doctrine connaît une forme de résurgence dans la conception du monde que véhicule la programmation orientée objet.

2 La programmation orientée objet

Le programmeur orienté objet agit comme le Dieu de Leibniz. Sa façon d'aborder une tâche est avant tout de se représenter les types d'objets — les

classes — que le programme fera interagir. Il lui faut concevoir l'ensemble des classes comme le Dieu de Leibniz crée dans son entendement l'ensemble des essences.

L'exemple que nous développerons ici sera le système d'information d'une bibliothèque. Le programmeur devra dans un premier temps distinguer au moins trois classes de choses : des adhérents, des œuvres et des exemplaires. Nous utiliserons ici le langage de programmation Java, mais les concepts que nous décrirons existent dans tous les langages orientés objets (Smalltalk, Eiffel, C++, etc.).

2.1 Les attributs

Une fois déterminées les classes de choses que le programmeur aura besoin de manipuler, celui-ci doit déterminer les propriétés de ces choses. Ces propriétés sont souvent appelées *attributs*. Ce sont par exemple :

1. pour un « adhérent » : un nom, un prénom, une date de naissance, une adresse, un état de l'abonnement (actif ou non), un numéro d'inscription ;

```
1   class Adherent {
2
3     // Nous expliquerons plus loin la
          signification du mot-clef private.
4
5     // Le numéro d'inscription est un nombre
          entier.
6     private int numeroInscription;
7
8     // Nom, prénom et adresse sont des chaînes de
          caractères.
9     private String nom;
10    private String prenom;
11    private String adresse;
12
13    // Le statut de la carte est une valeur de
14    // vérité (vrai ou faux).
15    private boolean carteActive;
16  }
```

2. pour une « œuvre » : un titre, un ou plusieurs auteurs, un éditeur, un numéro ISBN ;

```
1   class OEuvre {
2     private int identifiant; // Nombre entier.
3
4     // Chaînes de caractères.
5     private String titre;
6     private String editeur;
7     private String isbn;
8   }
```

3. pour un « exemplaire » : une cote, une indication de disponibilité (vrai ou faux), une indication d'état (« bon état », « abîmé »...), et surtout l'œuvre dont elle est l'exemplaire.

```
1   class Exemplaire {
2     private int identifiant; // Nombre entier.
3
4     OEuvre ouvrage; // L'œuvre dont voici un
          exemplaire.
5
6     // Chaînes de caractères.
7     private String cote;
8     private String etat;
9
10    // La disponibilité est une valeur de vérité.
11    private boolean disponibilite;
12  }
```

Une fois définies ces grandes classes, le programmeur devra parfois définir des sous-classes. Par exemple, une « œuvre » peut être un livre, une revue, etc. Supposons qu'un « livre » soit une œuvre possédant un auteur ; on définira alors la classe Livre de la façon suivante :

```
1  class Livre extends OEuvre {
2    // Outre tous les attributs standard d'une Œuvre,
3    // un Livre possède un auteur.
4    private String auteur;
5  }
```

Par le mécanisme d'*héritage simple*[3], disponible dans tous les langages de programmation orientée objet, ces classes et sous-classes s'organisent de manière hiérarchique. La structure des classes est alors arborescente, comme l'ordre des genres et des espèces chez Leibniz.

2.2 Les méthodes

De même qu'une monade leibnizienne est caractérisée non seulement par ses « qualités » mais également par ses « actions internes », chaque classe de la programmation orientée objet se caractérise non seulement par des « attributs » mais également par des « méthodes ».

Les *méthodes* sont les actes que les objets de cette classe peuvent effectuer. Un chien ne se caractérise pas seulement par une race, une taille et une robe, mais également par la faculté d'aboyer, de remuer la queue, de suivre son maître, de manger et de boire. Une classe se définit ainsi non seulement par ce que *sont* les objets mais tout autant par ce qu'ils *font*.

3. Certains langages, comme le C++ et Eiffel, autorisent également l'*héritage multiple*, c'est-à-dire la dérivation d'une classe à partir de plusieurs autres ; plusieurs langages orientés objets refusent cette option, source de nombreux problèmes.

Ainsi, définir ce qu'est l'adhérent d'une bibliothèque, ce n'est pas seulement énumérer ses propriétés caractéristiques (nom, adresse, etc.), mais aussi stipuler tout ce qu'il peut faire : recevoir du courrier, réserver un ouvrage, emprunter un exemplaire, se désinscrire, etc.

```
1   class Adherent {
2
3     // Méthodes
4
5     public void recevoirCourrier() {
6       ...
7     }
8
9     public void reserver(OEuvre ouvrage) {
10      ...
11    }
12
13    public void emprunter(Exemplaire reference) {
14      ...
15    }
16
17    public void desinscription() {
18      ...
19    }
20
21
22    // Attributs
23
24    private int numeroInscription;
25
26    private String nom;
27    private String prenom;
28    private String adresse;
29
30    private boolean carteActive;
31  }
```

2.3 Les objets

Une fois que le programmeur a créé toutes les classes par leurs attributs et méthodes — c'est-à-dire les « essences », définies par des « qualités » et des « actions internes » —, il peut créer les individus, appelés *objets*. Deux individus se distingueront toujours au minimum par leur adresse, c'est-à-dire par leur position dans l'espace de la mémoire ; mais ils se distinguent aussi généralement par leurs propriétés, typiquement par un identifiant numérique [4].

4. Dans l'ordinateur, deux individus se distinguent au minimum par leur adresse dans la mémoire, de même que deux individus du monde se distinguent au moins par leurs coordonnées spatio-temporelles. Mais Leibniz irait plus loin : selon son principe d'identité des indiscernables, deux individus différents se distingueront toujours par des propriétés

Afin de pouvoir créer de nouveaux individus, il nous faut ajouter à leur classe une méthode spéciale, que l'on appelle *constructeur*. C'est cette méthode qui « initialise » chaque nouvel individu, au moyen d'arguments donnés, par exemple un nom et un prénom.

```
1   class Adherent {
2
3     public Adherent(String nom, String prenom) {
4
5       // On donne à l'adhérent un nouveau numéro.
6       numeroInscription = dernierNumeroEnregistre + 1;
7
8       // On initialise son nom et son prénom à partir
9       // des arguments de la méthode.
10      this.nom = nom;
11      this.prenom = prenom;
12    }
13
14    ...
15  }
```

Ensuite, dans le cours du programme, on pourra à chaque fois créer un nouvel adhérent en spécifiant son nom et son prénom :

```
1   Adherent serge = new Adherent("Abiteboul", "Serge");
2   Adherent gilles = new Adherent("Dowek", "Gilles");
```

De même qu'une substance composée est constituée de monades, on peut définir des objets composés de plusieurs objets et possédant éventuellement leurs propres attributs et méthodes. Définissons par exemple un club d'adhérents comme un objet contenant une liste d'adhérents :

```
1   class Club {
2
3     // Méthode pour ajouter un adhérent au club.
4     public Adherent ajoutAdherent (Adherent adh) {
5       ...
6     }
7
8     // Attribut : liste d'adhérents.
9     private Adherent[] listeAdherents;
10    ...
11  }
```

Alors on peut créer un objet de la classe Club, en lui donnant pour constituants les objets de classes plus « atomiques » que sont ses adhérents :

---

et non seulement par leurs coordonnées.

> Il n'y a jamais dans la nature deux êtres qui soient parfaitement l'un comme l'autre, et où il ne soit possible de trouver une différence interne, ou fondée sur une dénomination intrinsèque. (ML 9)

Il n'existe pas, dans la nature, deux feuilles identiques.

```
1    Club binaire = new Club();
2
3    binaire.ajoutAdherent(serge);
4    binaire.ajoutAdherent(gilles);
```

On peut ainsi former des substances composées à partir de monades, et même des substances composées à partir de substances composées.

2.4 Les échanges de messages

Le programmeur connaît évidemment toutes les classes, car il en est l'unique auteur : « Dieu seul a une connaissance distincte de tout, car il en est la source » (PNG 13). Mais les objets eux-mêmes, comme les monades de Leibniz, n'ont pas cette omniscience. Tout au plus possèdent-ils des « perceptions confuses, qui enveloppent tout l'univers » (PNG 13) : ils ne connaissent le reste du monde que de façon biaisée, par l'action qu'il a sur eux, c'est-à-dire par l'ensemble des messages qu'ils reçoivent des autres substances et qu'ils leur envoient.

Les objets n'ont en effet, comme les monades, ni porte ni fenêtre : aucun objet ne peut agir directement sur un autre. Chaque objet possède, de par la définition de sa classe, une part privée (`private`) et une part publique (`public`). La part *privée* est l'ensemble des attributs et des méthodes auxquels il peut accéder de manière interne par le mot-clef `this`, et qui sont inaccessibles à tout autre objet ; il s'agit donc de son « apperception », du nom de la faculté que les informaticiens comme Leibniz appellent « réflexive ». La part *publique* est, quant à elle, l'ensemble des attributs et surtout des méthodes par lesquels les autres objets peuvent agir sur lui [5].

```
1    class Adherent {
2
3      // Seules des méthodes publiques permettent aux autres
4      // objets d'accéder au contenu des attributs.
5
6      public String getNom(){
7        return this.nom;
8      }
9
10     public void setNom(String nom){
11       this.nom = nom;
12     }
13     public afficheNom() {
14       System.out.print(this.nom);     // Affiche le nom.
15     }
16
17     public affichePrenom() {
18       System.out.print(this.prenom); // Affiche le prénom.
```

5. Souvent, on recommande de garder des attributs dans la sphère privée et de n'y autoriser l'accès que par des méthodes publiques.

```
19        }
20
21        ...
22
23        // Seul l'objet concerné a un accès direct à ses
24        // propres attributs.
25
26        private String nom;
27        private String prenom;
28        ...
29    }
```

L'action des autres objets sur un objet donné n'est pas immédiate : elle passe par l'envoi de messages. Chaque message est, du point de vue de l'objet qui l'envoie, l'expression d'une appétition, et, du point de vue de l'objet qui le reçoit, une perception.

Agir sur un autre objet, c'est lui intimer d'effectuer telle ou telle action interne. C'est lui demander de se modifier « spontanément », de l'intérieur. Aussi la programmation orientée objet requiert-elle, comme la monadologie de Leibniz, une harmonie préétablie entre les objets, garantissant que les messages soient pris en compte par le récepteur conformément à ce qu'en requiert et attend l'émetteur. L'harmonie préétablie est garantie par le Dieu qui a programmé le monde. Par cet échange incessant de messages, en programmation orientée objet comme dans la monadologie, « toute la nature est pleine de vie » (PNG 1).

3 Conclusion

Les métaphysiciens classiques — Descartes, Spinoza, Leibniz, Malebranche — ont reconnu l'intérêt philosophique des sciences formelles en s'inspirant essentiellement des concepts et méthodes des mathématiques. C'est parce qu'ils ne connaissaient pas l'informatique, où ils auraient trouvé une riche source d'inspiration.

L'exemple de la programmation orientée objet montre que les styles de programmation peuvent exprimer des conceptions du monde à part entière, comparables sur plus d'un point avec celles que les métaphysiciens s'efforcent de construire. Plus qu'une occupation technique, la programmation est une activité conceptuelle dans laquelle l'homme impose à la machine ses propres modes de représentation. Styles de programmation et systèmes de métaphysique décrivent ainsi des ontologies dont les propriétés peuvent être analysées selon les mêmes méthodes.

4 Bibliographie

Pour aller plus loin, lire Jean-François Perrot, « Objets, classes et héritage : définitions » (`http://www-poleia.lip6.fr/~briot/colloque-JFP/revue/textes/chap1-book-cimpa.pdf`), in Roland Ducournau et alii (éd.), *Langages et modèles à objets : état des recherches et perspectives*, collection Didactique, INRIA, 1998 (`https://hal.archives-ouvertes.fr/inria-00340768/document`), p. 3–31.