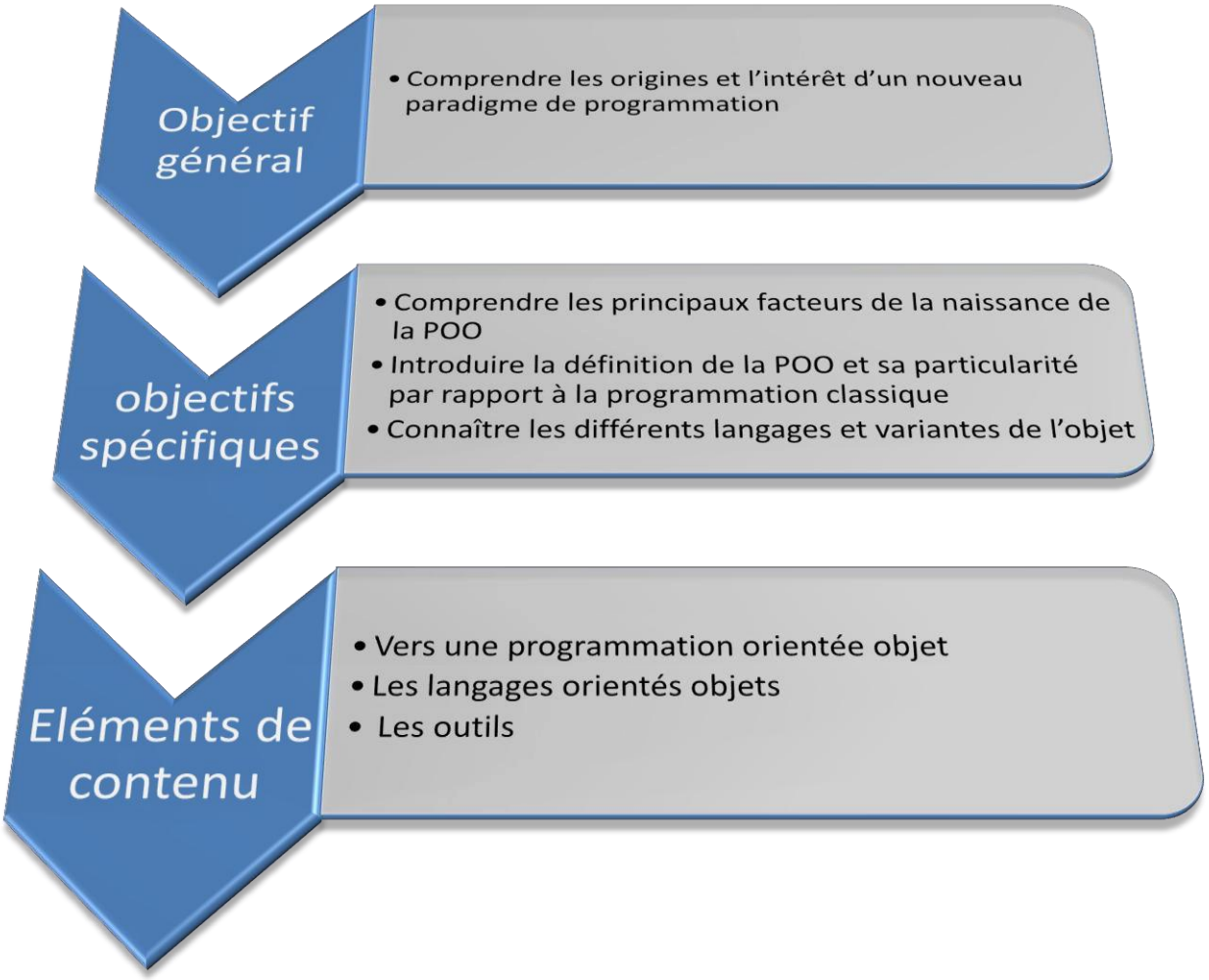
Chapitre 1 : Introduction à la programmation orientée objet



Introduction

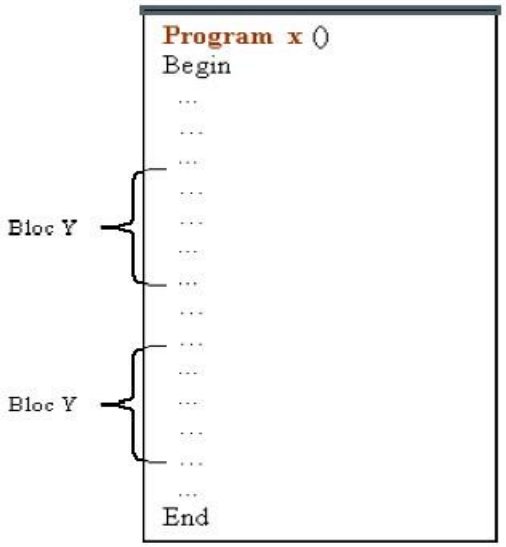
En termes de programmation, il y a eu plusieurs évolutions successives. Parmi lesquelles on peut citer la programmation procédurale et la programmation orientée objet.

Pour voir quel cheminement le programmeur a dû suivre pendant ces dernières années, on propose de mettre en parallèle ces deux techniques de programmation puis de mettre l'accent sur la programmation orientée objet, sur quelques langages orientés objets et environnements de développements intégrés.

I. Vers une programmation orientée objet

1.1 La programmation structurée

Il s'agit de programmes simples et circonscrits ne nécessitant pas de décomposition en sous-programmes. Dans ce cas de figure, le programme principal est conçu pour que les instructions manipulent des variables globales, dont les valeurs sont définies et modifiées.

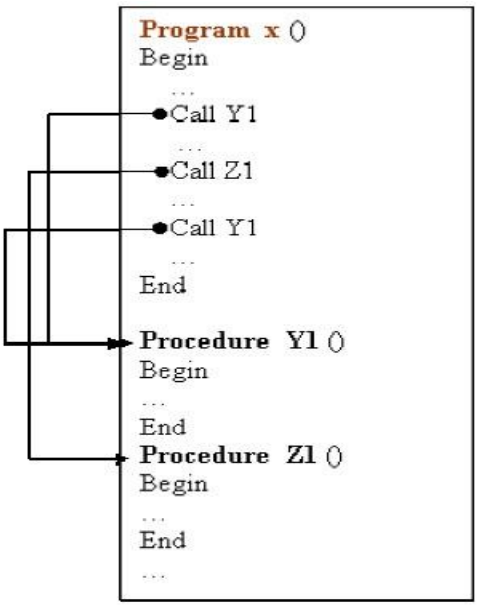


Inconvénients : ce mode de programmation implique que les instructions de traitement doivent être réécrites à chaque fois qu'on aura besoin.

1.2 La programmation procédurale

Les instructions sont rassemblées dans ce qu'il est convenu d'appeler des procédures et utilisées sans que le code ne soit réécrit plusieurs fois. Lorsqu'une procédure est appelée, les instructions qu'elle contient sont exécutées, puis le programme retourne à l'endroit suivant l'appel de la procédure.

L'introduction de paramètres et l'imbrication des procédures permettent d'améliorer la structuration des programmes. Par ailleurs, la proportion d'erreurs est plus réduite.

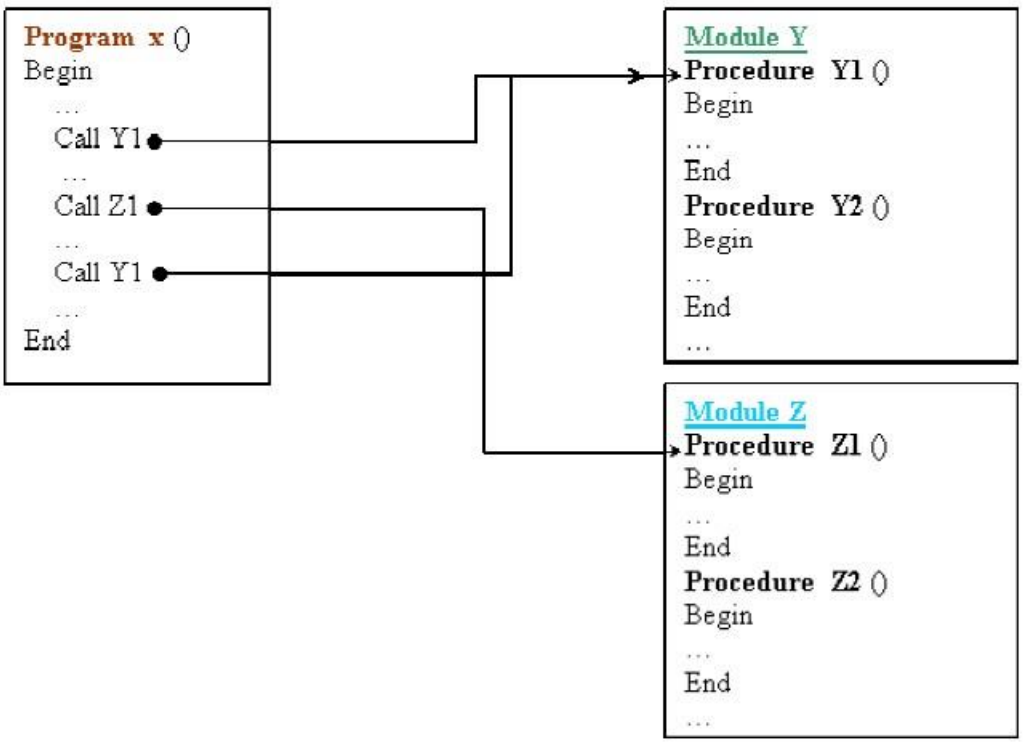


L'utilisation séparée des procédures conduit à subdiviser à priori les tâches à résoudre en sous-tâches et à regrouper en modules les procédures requises, d'où la programmation modulaire

1.3 La programmation modulaire

Cette technique vient pour assurer une meilleure structuration du programme en le divisant en blocs logiques contenant des données et des procédures, appelés des modules. L'idée est que, dans un programme, à chaque tâche peut correspondre un module et les opérations de traitement à réaliser au sein de ce module peuvent être regroupées dans des procédures.

Le but est de minimiser l'interdépendance entre les différentes parties du programme, afin de faciliter le changement de chacune d'elles sans affecter le reste du programme et de pouvoir réutiliser ces modules (les modules sont compilés séparément).



1.4 La programmation orientée objet (POO)

La POO est une nouvelle méthode de programmation qui se distingue fondamentalement des méthodes précédentes par ses trois principes qui sont l'encapsulation, l'héritage et le polymorphisme. Elle tend à se rapprocher de notre manière naturelle d'appréhender le monde. L'approche Orientée Objet s'est surtout posé la question« **sur quoi porte le programme** ? ».

En effet, un programme informatique comporte toujours des traitements mais aussi, et surtout des données. Si la programmation procédurale s'intéresse aux traitements puis aux données, la POO s'intéresse d'abord aux données, auxquelles elle associe ensuite des traitements : l'expérience a montré que les données sont ce qu'il y a de plus stable dans la vie d'un programme, il est donc intéressant d'architecturer le programme autour de ses données.

Ainsi, la technique de POO repose sur le postulat suivant : « la meilleure façon de décrire les systèmes informatiques est de se fonder, non pas sur ce qu'ils font, mais sur les objets qu'ils manipulent. »

Exemple :

Un programme de comptabilité sera bâti, non pas autour de sa fonction principale, mais autour des objets qu'il traite : comptes, états, écritures, …

La résolution d'un problème consiste ainsi en une interaction entre différents objets. La subdivision classique d'un programme en données et instructions de traitement est radicalement modifiée avec la POO. Elle est supplantée par la constitution d'objets synthétisant les données et les traitements effectués sur elles. La résolution d'un problème fait exclusivement appel à des messages échangés entre ces objets.

1.5Approche par traitements vs approche par objets

1.5.1 Exemple

Prenons l'exemple d'une application de commerce électronique. Dans un système procédural, on trouverait les procédures suivantes:

- Créer les produits
- Classer les produits
- Modifier les produits
- Afficher les produits
- Ajouter les produits au panier
- Acheter les produits

Dans un système **orienté objet** l'application entière pourrait ne fonctionner qu'avec quelques objets, chacun responsable d'un aspect précis du système:

- Produit
- Client
- Panier
- Entrepôt

1.5.2 Tableau de comparaison : Approche par Traitements/Approche par Objets

Le tableau suivant permet de comparer les deux approches de programmation par traitements d'une part et par objets d'autre part.

Approche par Traitements	Approche par Objets
« Que doit faire le programme ? » **Principe**: Analyse fonctionnelle descendante qui consiste à diviser la tâche à accomplir en sous-tâches moins complexes.	**« Sur quoi porte le programme ? »** **Principe:** Pour écrire un programme, on se fonde non pas sur ce qu'il fait, mais sur les objets qu'ils manipulent auxquels on associe des traitements. **Avantages : la POO privilégie :**

Avantages: facile à programmer car elle fait appel à un raisonnement logique que chacun sait faire naturellement. **Problèmes**: Ne favorise pas les critères de : • Réutilisabilité : les procédures ne sont pas générales, mais adaptées aux sous problèmes pour lesquels elles ont été conçues. • Extensibilité: l'architecture du logiciel est fondée sur les traitements, or ces derniers sont moins stables que les données. D'où cette approche est inadaptée à la conception de gros logiciels.	• La réutilisabilité : un composant logiciel doit être organisé autour d'une structure de données complète et non pas seulement assurer une fonction unique, aussi importante soit-elle. En d'autres termes ce ne sont pas des programmes qu'il faut pouvoir réutiliser, mais bien plutôt des structures de données complètes. • L'extensibilité : L'intérêt des méthodes orientée objet découle ici d'une remarque empirique : si l'on observe l'évolution d'un logiciel d'une certaine taille, on constate que les fonctions demandées au système changent tout au long de sa vie active; les objets auxquels s'appliquent ces fonctions, en revanche restent plus stables. Pour préserver la souplesse et l'extensibilité du système, mieux vaut bâtir sa structure sur ses caractéristiques les plus durables.

II. Les langages orientés objets

2.1 Naissance de la notion d'objet

La notion d'objet date du projet du missile MINUTEMAN en 1957. La conception et la simulation du fonctionnement de ce missile reposait sur un ensemble de composants logiciels prenant en charge la partie physique du missile, les trajectoires, les différentes parties du vol,…

Le fonctionnement du système global reposait sur l'échange de messages d'information entre les différents composants. Chaque composant logiciel était conçu par un spécialiste et possédait ses données privées. De même, le composant était virtuellement isolé du reste du programme par l'ensemble de ses méthodes servant d'interface.

2.2 Histoire des langages orientés objets

Les méthodes et langages à objets ne sont pas une invention aussi récente qu'on pourrait le croire.

- La naissance de ces concepts est survenue avec le langage **Simula** en 1967 conçu par Ole-Johan Dahl et Krysten Nygaard. Ce langage comprenait les éléments principaux de la méthode, mais il venait trop tôt : c'est avec l'un de ses successeurs, **Smalltalk,** qu'un large public a pris connaissance 15 ans plus tard des idées introduites par Simula.
- Avec **Smalltalk** la combinaison des techniques à objets, d'un environnement de développement personnel puissant et d'un support graphique de très haute qualité, a rendu soudain la méthode des objets attrayante et tangible.
- D'autres langages sont apparus ces dernières années. Certains sont de simples extensions de langages existants (langages hybrides) : on trouve en particulier les successeurs du C, **C++** et **objective-C**, qui ajoutent à C quelques unes des idées de la programmation par objets, mais sans remettre en cause la structure du langage de base.
- Apple a développé **Object Pascal**. De nombreuses extensions « à objets » de Lisp ont été également proposées, telles que **Loops** (de Xérox), **Flavors** (du MIT) et **Ceyx** (de l'INRIA).
- Les langages **Modula-2** et **Ada** ont également subi l'influence de Simula
- Le langage **Eiffel** est un langage autonome écrit en C purement orienté objet. Il a été conçu par Bertrand Meyer pour les applications scientifiques et de gestion. Eiffel est considéré comme l'un des plus purs langages à objets car il est développé en suivant de près la philosophie objet.
- Le développement de **Java** a commencé en 1990 chez SUN Microsystems sous la direction de James Gosling. Il est dérivé du C++ et permet de pallier à ses inconvénients (pointeurs, surcharge d'opérateurs, héritage multiple, gestion de la mémoire, etc.). Il est indépendant de la plate-forme. Il s'exécute sous n'importe quelle plate-forme pour autant que celle-ci possède un interpréteur Java.
- **C#** est un langage orienté objet développé par **Microsoft** pour permettre la création d'une large gamme d'applications. Ce langage est conçu pour offrir un développement rapide au programmeur C++ sans sacrifier les qualités de puissance et de contrôle qui sont la marque du C et du C++.

III. Les IDE : Integrated Develompent Environment

Un environnement de développement est un ensemble d'outils qui, en plus des tâches classiques d'un langage de programmation offre des fonctionnalités étendues permettant de couvrir une plus large partie du cycle de création d'un logiciel.

On peut distinguer des tâches telles que :

- Compilation et débogage
- Déploiement de l'application.
- Outils de test et de vérification.
- Outil de création graphique (icône).
- Interfaçage avec les SGBD.
- Générateur de menus.
- Un ensemble d'assistants.

3.1 Exemples d'IDE

Dans ce qui suit, des exemples d'IDE chez des éditeurs différents et basés sur des langages de programmation différents :

- IntelliJ Idea
- Borland JBuilder
- Oracle JDeveloppper
- Visual J++
- UltraEdit
- Eclipse
- NetBeans…..

QCM

Répondez aux questions en cochant la réponse.

1. Quel est le type de programmation le plus ancien ?
   - ☐ Programmation orientée objet
   - ☐ Programmation procédurale
   - ☐ Programmation modulaire
2. La programmation modulaire est basée sur
   - ☐ modules
   - ☐ procédures
   - ☐ objets
3. un module est un ensemble de
   - ☐ procédure
   - ☐ sous programmes et des données
   - ☐ programmes
4. La meilleure façon de décrire les systèmes informatiques est de se fonder sur :
   - ☐ ce qu'ils font
   - ☐ les modules qu'ils manipulent
   - ☐ les objets qu'ils manipulent.
5. L'expérience a montré que :
   - ☐ les traitements sont ce qu'il y a de plus stable dans la vie d'un programme.
   - ☐ les données sont ce qu'il y a de plus stable dans la vie d'un programme.
   - ☐ il est intéressant d'architecturer le programme autour de ses traitements.

5. Quel est le plus ancien des langages orientés objets ?
   - ☐ SmallTalk
   - ☐ C++
   - ☐ Simula
6. Les langages objets ont été inventés
   - ☐ Au début des années 70
   - ☐ A la fin des années 70
   - ☐ A la fin des années 60
7. Java
   - ☐ Est un langage interprété
   - ☐ Est un langage compilé
   - ☐ Est un langage interprété et compilé
8. JRE est l'acronyme de
   - ☐ Java Run Time Execution
   - ☐ Java Run Time Environement
   - ☐ Java RunTime Entreprise
9. Sur la machine où doit s'exécuter un programme Java, nous avons besoin de
   - ☐ JDK+JRE
   - ☐ JRE seulement
   - ☐ JDK seulement