Glossaire Informatique

Acronymes

API
: *Application Programming Interface.*

CLI
: *Command Line Interface.*

DLL
: *Dynamic Link Library.*

DSS
: Diagramme de Séquence Système.

EDI
: Environnement de Développement Intégré.

GNU
: *GNU's Not UNIX.*

GPL
: *GNU General Public License.*

GUI
: *Graphical User Interface.*

IDE
Integrated Development Environment.

IHM
Interface Homme-Machine.

JDK
Java Development Kit.

OS
Operating System.

PHP
PHP: *Hypertext Preprocessor*.

POO
Programmation Orientée Objet.

POSIX
Portable Operating System Interface.

SDK
Software Development Kit.

SE
Système d'Exploitation.

SGBD
Système de Gestion de Bases de Données.

SGBDR
Système de Gestion de Bases de Données Relationnelles.

SO
Shared Object.

SQL
Structured Query Language.

UI
User Interface.

UML
Unified Modeling Language.

UP
Unified Process.

Programmation

algorithme
Un algorithme énonce une résolution d'un problème sous la forme d'une série d'opérations à effectuer. Un algorithme est une suite finie d'instructions élémentaires écrites dans un langage universel et exécutées de manière séquentielle. La mise en œuvre de l'algorithme consiste en l'écriture de ces opérations dans un langage de programmation et constitue alors la brique de base d'un programme informatique. Un algorithme doit être suffisamment général pour permettre de traiter une classe de problèmes. Pour un problème donné, il peut y avoir un ou plusieurs algorithmes ou aucun.

bibliothèque logicielle
Une bibliothèque logicielle est un ensemble de fonctions utilitaires, regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire. Les bibliothèques logicielles ne sont pas

complètement des « exécutables » car elles ne possèdent pas de programme principal et par conséquent ne peuvent pas être exécutées directement. Les bibliothèques logicielles sont : une interface de programmation (*Application Programming Interface* (API)), les composants d'un kit de développement logiciel (*Software Development Kit* (SDK)) parfois regroupées en un *framework*, de façon à constituer un ensemble cohérent et complémentaire de bibliothèques. Elles sont des fichiers : `.DLL` (*Dynamic Link Library*) ou Bibliothèque de liens dynamiques sous Windows et `.so` (*Shared Object*) ou Bibliothèque dynamique sous GNU/Linux. On retrouve parfois le terme impropre de « librairie » issu du faux ami anglais *library*.

fonction
Une fonction est un sous-programme (sous-routine) qui retourne une et une seule valeur. Elle ne permet de ne récupérer qu'un résultat. Par convention, ce type de sous-programme ne devrait pas interagir avec l'environnement (écran, utilisateur). Dans certains langages de programmation, les fonctions permettent d'écrire aussi des procédures.

framework
Un *framework* est constitué d'un ensemble cohérent et complémentaire de bibliothèques. Il fournit un cadre de développement. Les *frameworks* sont conçus et utilisés pour modeler l'architecture des logiciels applicatifs, des applications web, des composants logiciels.

implémentation (codage)
Les informaticiens utilisent fréquemment l'anglicisme implémentation pour désigner l'écriture d'un algorithme dans un langage de programmation. L'écriture en langage informatique est aussi fréquemment désignée par le terme « codage », qui n'a ici aucun rapport avec la cryptographie, mais qui se réfère au terme « code source » pour désigner le texte, en langage de programmation, constituant le programme.

instruction
Les langages de programmation offrent le concept d'instructions dans lesquelles on pourra utiliser des opérateurs. On rappelle que les instructions que l'ordinateur peut comprendre ne sont pas celles du langage humain. Le processeur sait juste exécuter un nombre limité d'instructions bien définies (son jeu d'instructions). La première des instructions est la possibilité d'affecter une valeur à une variable. La plupart des langages proposent l'opérateur d'affectation `=` pour cela.

paradigme
Un paradigme est un style fondamental de programmation, définissant la manière dont les programmes doivent être formulés. Les plus connus sont : programmation impérative, programmation orientée objet, programmation procédurale, programmation structurée, programmation concurrente, ...

paramètre
Un paramètre est une donnée manipulée par une fonction (ou méthode) et connue du code appelant cette fonction. On distingue deux types de paramètres : les paramètres d'entrée et les paramètres de sortie. Un paramètre d'entrée est une donnée fournie par le code appelant au code appelé. Cette donnée peut être transmise de deux façons : passage par copie (aussi appelé par valeur) : le code appelé dispose d'une copie de la valeur qu'il peut modifier sans affecter l'information initiale dans le code appelant et passage par adresse ou par référence : le code appelé dispose d'une information lui permettant d'accéder en mémoire à la valeur que le code appelant cherche à lui transmettre. Il peut alors modifier cette valeur là où elle se trouve ; le code appelant aura accès aux modifications faites sur la valeur. Dans ce cas, le paramètre peut aussi être utilisé comme un paramètre de sortie. Un paramètre de sortie est une donnée fournie par le code appelé au code appelant.

procédure
Une procédure est un sous-programme (ou sous-routine) qui permet de récupérer de 0 à n résultats. Par convention, ce type de sous-programme peut interagir avec l'environnement (écran, utilisateur).

La notion de procédure ne se retrouve pas dans tous les langages de programmation. En C/C++, les procédures sont implémentables par des fonctions.

programmation

La programmation est l'ensemble des activités qui permettent l'écriture des programmes informatiques. C'est une étape importante du développement de logiciels. La programmation représente la rédaction du code source d'un logiciel. On utilise plutôt le terme développement pour dénoter l'ensemble des activités liées à la création d'un logiciel : cela inclut la spécification du logiciel, sa conception, puis son implémentation proprement dite au sens de l'écriture des programmes dans un langage de programmation bien défini et aussi la vérification de sa correction ....

programmation concurrente

La programmation concurrente est un paradigme de programmation tenant compte, dans un programme, de l'existence de plusieurs fils d'exécution qui peuvent être appelés threads, processus ou tâches. La programmation concurrente est plus complexe et difficile que la programmation impérative.

programmation impérative

La programmation impérative est un paradigme de programmation qui décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. Ce type de programmation est le plus répandu parmi l'ensemble des langages de programmation existants. La quasi-totalité des processeurs qui équipent les ordinateurs sont de nature impérative : ils sont faits pour exécuter une suite d'instructions élémentaires. Les langages de plus haut niveau utilisent des variables et des opérations plus complexes, mais suivent le même paradigme. La grande majorité des langages de programmation est impérative. La plupart des langages de haut niveau comporte cinq types d'instructions principales : la séquence d'instructions, l'assignation ou affectation, l'instruction conditionnelle, la boucle et les branchements.

programmation orientée objet

La programmation orientée objet (POO) consiste à définir des objets logiciels et à les faire interagir entre eux. Il existe actuellement deux grandes catégories de langages à objets : les langages à classes (C++, C#, Java, Python, ...) et les langages à prototypes (JavaScript).

programmation procédurale

La programmation procédurale est un paradigme qui se fonde sur le concept d'appel procédural. Une procédure, aussi appelée routine, sous-routine ou fonction, contient simplement une série d'étapes à réaliser. N'importe quelle procédure peut être appelée à n'importe quelle étape de l'exécution du programme, y compris à l'intérieur d'autres procédures, voire dans la procédure elle-même (récursivité). Un programme est donc composé de procédures (et de fonctions) interagissant entre elles.

programmation structurée

La programmation structurée, apparue en 1970, constitue un sous-ensemble de la programmation impérative. Elle est en effet célèbre pour son essai de suppression de l'instruction `goto` ou du moins pour la limitation de son usage à des cas exceptionnels. La programmation structurée recommande une organisation hiérarchique simple du code. Les programmeurs décomposent leur code en modules (appelés fonctions et procédures dans certains langages) ne dépassant guère 60 lignes, afin d'être présente en entier sous les yeux. On recommande aux programmes d'éviter l'usage des variables globales afin de prévenir les effets de bord (*side effects*). La plupart des programmeurs ont adopté la programmation structurée. Dijkstra rappelait qu'un programme devait d'abord être compris par le programmeur et ses collègues chargés de la maintenance, et que si cette tâche était accomplie, le reste (le faire exécuter à la machine) n'était plus que formalité.

routine

Une routine est une entité informatique qui encapsule une portion de code (une séquence d'instructions) effectuant un traitement spécifique bien identifié (asservissement, tâche, calcul, etc.) relativement indépendant du reste du programme, et qui peut être réutilisé dans le même pro-

gramme, ou dans un autre. Dans ce cas, on range souvent la routine dans une bibliothèque pour la rendre disponible à d'autres projets de programmation, tout en préservant l'intégrité de son implémentation. En programmation informatique, on retrouve les routines sous deux formes principales : la procédure et la fonction.

variable
Pour manipuler des données, les langages de programmation offre le concept de variable. Une variable est un espace de stockage pour un résultat. Dans un programme, une variable est associé à un symbole (habituellement un nom qui sert d'identifiant) qui renvoie à une position de la mémoire (une adresse) dont le contenu peut prendre successivement différentes valeurs pendant l'exécution d'un programme. De manière générale, les variables ont un type : c'est la convention d'interprétation de la séquence de bits qui constitue la variable. Le type de la variable spécifie aussi sa taille (la longueur de cette séquence) soit habituellement 8 bits, 32 bits, 64 bits, ... Cela implique qu'il y a un domaine de valeurs (ensemble des valeurs possibles). Et elles ont aussi une valeur : c'est la séquence de bits elle même. Cette séquence peut être codée de différrentes façons suivant son type.

Langages de programmation

Assembleur
C'est un langage de bas niveau qui représente le langage machine sous une forme lisible par un humain. Les combinaisons de bits du langage machine sont représentées par des mnémoniques (des symboles) faciles à retenir : `ADD` pour l'addition, `MOV` pour la copie de valeurs, etc ....

C
C est un langage de programmation impératif et généraliste. Inventé au début des années 1970 pour réécrire UNIX, C est devenu un des langages les plus utilisés. Il a été développé en même temps que UNIX par Dennis Ritchie et Ken Thompson.

C#
C# est un langage de programmation orienté objet, commercialisé par Microsoft depuis 2002 et destiné à développer sur la plateforme Microsoft .NET. Il est dérivé du C++ et très proche du Java dont il reprend la syntaxe générale ainsi que les concepts, y ajoutant des notions telles que la surcharge des opérateurs, les indexeurs et les délégués.

C++
C++ est un langage de programmation compilé créé par Bjarne Stroustrup dans les années 1980. On peut considérer que C++ « est du C » avec un ajout de fonctionnalités dont la notion de classes. Le langage C++ utilise les concepts de la programmation orientée objet.

Java
Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, en 1995. La particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Mac OS, Windows ou Linux, avec peu ou pas de modifications.

JavaScript
JavaScript est un langage de programmation de scripts, créé en 1995 par Brendan Eich, principalement employé dans les pages web interactives mais aussi pour les serveurs (par exemple Node.js). Il a été standardisé sous le nom d'ECMAScript. C'est un langage orienté objet à prototype. Le langage supporte le paradigme objet, impératif et fonctionnel.

Langage de programmation
Un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes

et produire des programmes informatiques. Quel que soit le langage évolué utilisé, il faudra le traduire en langage machine. Il existe des approches différentes : l'assemblage (assembleur) pour une traduction du mnémonique (symbole) en *opcode*, la compilation (C, C++, ...) pour une traduction du code source en langage machine (code objet), l'interprétation (Javascript, ...) pour une traduction « à la volée » de chaque instruction avant de l'exécuter, la compilation et interprétation (Java) pour une compilation en un code intermédiaire (*bytecode*) qui n'est pas du code machine mais un code pour une « machine virtuelle » puis ce code est interprété par un interpréteur (la machine virtuelle).

Langage machine

Le langage machine est le seul langage que le processeur puisse traiter. Il est composé d'instructions et de données à traiter codées en binaire. Un processeur possède nativement un jeu d'instructions composé d'*opcode* (code opération) et d'opérandes. Il existe des *opcodes* pour faire des opérations arithmétiques, logiques, etc ... Chaque instruction nécessite un certain temps (généralement un nombre de cycles d'horloge) pour s'exécuter.

Perl

Perl est un langage de programmation créé par Larry Wall en 1987 pour traiter facilement de l'information de type textuel. Ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C. Le langage Perl est sous licence libre *GNU General Public License* (GPL) et il est multi-plate-forme. Le langage est apprécié des administrateurs système.

PHP

PHP: *Hypertext Preprocessor* (PHP) est un langage de programmation libre, créé en 1994 par Rasmus Lerdorf, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet.

Python

Python est un langage de programmation objet, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions. Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Programmation Orientée Objet

amitié

L'amitié en C++ permet à des fonctions, des méthodes ou des classes d'accéder à des membres privées d'une autre classe. Les amies se déclarent en faisant précéder la déclaration classique du mot clé `friend` à l'intérieur de la déclaration de la classe cible. L'amitié n'est pas transitive : les amis des amis ne sont pas des amis. Une classe A amie d'une classe B, elle-même amie d'une classe C, n'est pas amie de la classe C par défaut. L'amitié n'est pas héritée : mes amis ne sont pas les amis de mes enfants. Si une classe A est amie d'une classe B et que la classe C est une classe fille de la classe B, alors A n'est pas amie de la classe C par défaut.

attribut

Un attribut est une variable membre d'un objet.

classe

Une classe décrit des propriétés communes (caractéristiques, responsabilités, comportements) d'un ensemble d'objets. Elle apparaît comme un type (ou un « moule à objets ») à partir duquel il sera possible de créer des objets. Les éléments de cet ensemble sont les instances de la classe (les objets).

classe abstraite
Une classe est dite abstraite lorsqu'on ne peut instancier d'objets à partir de cette classe. En C++, cela signifie qu'elle contient au moins une méthode déclarée virtuelle pure.

constructeur
Lorsqu'on instancie un objet, le constructeur de la classe est appelé automatiquement au moment de la création. Un constructeur est chargé d'initialiser l'état d'un objet (donc ses membres). Un constructeur est une méthode (ou plusieurs) qui porte toujours le même nom que la classe. Il peut avoir des paramètres, et des valeurs par défaut. Il n'a jamais de type de retour.

constructeur de copie
Le constructeur de copie permet d'initialiser un nouvel objet à partir d'un objet existant. Il est appelé dans la création d'un objet à partir d'un autre objet pris comme modèle, le passage en paramètre d'un objet par valeur à une fonction ou une méthode et le retour d'une fonction ou une méthode renvoyant un objet.

constructeur par défaut
Un constructeur par défaut a pour rôle de créer une instance non initialisée quand aucun autre constructeur fourni n'est applicable. Il est donc conseillé de toujours écrire un constructeur par défaut.

destructeur
Le destructeur est la méthode membre appelée automatiquement lorsqu'une instance (objet) de classe cesse d'exister en mémoire. Son rôle est de libérer toutes les ressources qui ont été acquises par cet objet. Un destructeur est une méthode qui porte toujours le même nom que la classe, précédé de "~". Il ne possède aucun paramètre. Il n'y en a qu'un et un seul. Il n'a jamais de type de retour.

encapsulation
L'encapsulation est l'idée de protéger les variables contenues (les attributs) dans un objet et de ne proposer que des méthodes pour les manipuler. En respectant ce principe, tous les attributs d'une classe seront donc privées (`private`). L'objet est ainsi vu de l'extérieur comme une « boîte noire » possédant certaines propriétés et ayant un comportement spécifié. C'est le comportement d'un objet qui modifiera son état. On ajoutera souvent (mais pas obligatoirement) des méthodes publiques pour établir un accès contrôlé aux attributs privés de la classe : un accesseur `get()` (*getter*) qui permet l'accès en lecture et un mutateur ou manipulateur `set()` (*setter*) qui permet l'accès en écriture.

exception
Les exceptions ont été ajoutées à la norme du C++ afin de faciliter la mise en œuvre de code robuste. Une exception est l'interruption de l'exécution du programme à la suite d'un événement particulier (= exceptionnel !) et le transfert du contrôle à des fonctions spéciales appelées gestionnaires. Le but des exceptions est de réaliser des traitements spécifiques aux événements qui en sont la cause. La gestion d'une exception est découpée en deux parties distinctes : le déclenchement avec l'instruction `throw` et le traitement (l'inspection et la capture) avec deux instructions inséparables `try` et `catch`.

héritage
L'héritage est un concept fondamental de la programmation orientée objet. Il se nomme ainsi car le principe est en quelque sorte le même que celui d'un arbre généalogique. Ce principe est fondé sur des classes « filles » qui héritent des caractéristiques des classses « mères ». L'héritage permet d'ajouter des propriétés à une classe existante pour en obtenir une nouvelle plus précise. Il permet donc la spécialisation ou la dérivation de types. En utilisant l'héritage, il est possible d'ajouter des caractéristiques (attributs) et/ou des comportements propres (méthodes), d'utiliser les caractéristiques héritées et de redéfinir les comportements (méthodes héritées).

instance
Une instance d'une classe est un objet. Instancier revient à créer des objets à partir de classes.

liste d'initialisation
La liste d'initialisation permet d'utiliser le constructeur de chaque donnée membre, et ainsi d'éviter une affectation après coup. La liste d'initialisation doit être utilisée pour certains objets qui ne peuvent pas être contruits par défaut : c'est le cas des références et des objets constants. La liste d'initialisation est aussi utilisée pour appeler le constructeur d'une classe parente en cas d'héritage. Elle suit la définition du constructeur après avoir ajouté le caractère '`:`'. Chaque initialisation est séparée par des virgules '`,`'.

membre statique
Un membre statique est déclaré avec l'attribut `static` et il sera partagé par tous les objets de la même classe. Il existe même lorsque aucun objet de cette classe n'a été créé. Un membre statique peut être un attribut ou une méthode. Un attribut statique doit être initialisé explicitement, à l'extérieur de la classe (même s'il est privé), en utilisant l'opérateur de résolution de portée (::) pour spécifier sa classe. En général, son initialisation se fait dans le fichier `.cpp` de définition de la classe. On peut accéder directement à un membre statiquer en le nom de la classe suivi de l'opérateur de résolution de portée :: et du nom du membre.

méthode
Une méthode est une fonction membre d'un objet.

méthode constante
On déclare et on définit une méthode constante en ajoutant le modificateur `const`. Une méthode constante est tout simplement une méthode qui ne modifie aucun des attributs de l'objet.

méthode inline
C++ présente une amélioration des macros du langage C avec les fonctions (et donc les méthodes) *inline*. Elles ont le comportement des fonctions (vérification des arguments et de la valeur de retour) et elles sont substituées dans le code après vérification. Une méthode est dite *inline* lorsque le corps de la méthode est définie directement dans la déclaration de la classe ou lorsqu'on définit une méthode, on ajoute au début de sa définition le mot-clé `inline`. Les fonctions *inline* (ou les macros) ont l'avantage d'être plus rapide qu'une fonction normale mais les inconvénients de produire un programme binaire plus volumineux et de compliquer la compilation séparée.

méthode virtuelle
Une méthode virtuelle est une méthode définie dans une classe qui est destinée à être redéfinie dans les classes qui en héritent. En C++, on utilisera le mot clé `virtual` dans la déclaration de la méthode. Les fonctions virtuelles permettent d'exprimer des différences de comportement entre des classes de la même famille. Ces différences sont ce qui engendre un comportement polymorphe.

méthode virtuelle pure
Une méthode virtuelle ne possédant qu'une déclaration à l'intérieur d'une classe, sans être définie dans cette classe est dite abstraite ou virtuelle pure. Dans ce cas, la classe est dite abstraite car il sera impossible techniquement d'instancier un objet à partir de cette classe. En C++, la syntaxe spéciale `= 0` est utilisée dans la déclaration de la méthode.

objet
Concrètement, un objet est une structure de données (ses attributs c.-à-d. des variables) qui définit son état et une interface (ses méthodes c.-à-d. des fonctions) qui définit son comportement. Un objet est créé à partir d'un modèle appelé classe. Chaque objet créé à partir de cette classe est une instance de la classe en question. Un objet posséde une identité qui permet de distinguer un objet d'un autre objet (son nom, une adresse mémoire).

objet constant
On déclare un objet constant avec le modificateur `const`. On ne pourra pas modifier l'état (ses

attributs) d'un objet constant. On ne peut appliquer que des méthodes constantes sur un objet constant.

paramètre par défaut
Le langage C++ offre la possibilité d'avoir des valeurs par défaut pour les paramètres d'une fonction (ou d'une méthode), qui peuvent alors être sous-entendus au moment de l'appel. Il suffit de faire suivre la déclaration du paramètre de l'opérateur `=` et d'une valeur.

polymorphisme
Le polymorphisme est un moyen de manipuler des objets hétéroclites de la même manière, pourvu qu'ils disposent d'une interface commune. Un objet polymorphe est donc un objet susceptible de prendre plusieurs formes pendant l'exécution. Le polymorphisme représente la capacité du système à choisir dynamiquement la méthode qui correspond au type de l'objet en cours de manipulation. Le polymorphisme est implémenté en C++ avec les méthodes virtuelles (`virtual`) et l'héritage.

programmation orientée objet
La programmation orientée objet (POO) consiste à définir des objets logiciels et à les faire interagir entre eux. Il existe actuellement deux grandes catégories de langages à objets : les langages à classes (C++, C#, Java, Python, ...) et les langages à prototypes (JavaScript).

redéfinition (*overriding*)
Une redéfinition (*overriding*) permet de fournir une nouvelle définition d'une méthode d'une classe ascendante pour la remplacer dans le cas d'un héritage. Elle doit avoir une signature rigoureusement identique à la méthode parente. La signature d'une méthode (ou d'une fonction) comprend son nom et sa liste de paramètres. Le type de retour ne fait pas partie de la signature.

surcharge (*overloading*)
Une surcharge (ou surdéfinition) permet d'utiliser plusieurs méthodes qui portent le même nom au sein d'une même classe avec des signatures différentes. La signature d'une méthode (ou d'une fonction) comprend son nom et sa liste de paramètres. Le type de retour ne fait pas partie de la signature. Pour surcharger une méthode, il suffit que le type et/ou le nombre de paramètres soit différent.

template
En C++, la fonctionnalité template fournit un moyen de réutiliser du code source. Les *templates* permettent d'écrire des fonctions et des classes en paramétrant le type de certains de leurs constituants (type des paramètres ou type de retour pour une fonction, type des éléments pour une classe collection par exemple). Les *templates* permettent d'écrire du code générique, c'est-à-dire qui peut servir pour une famille de fonctions ou de classes qui ne diffèrent que par la valeur de ces paramètres.

visibilité
La notion de visibilité indique qui peut avoir accès à un membre. La visibilité est précisée au sein des classes (et non des objets) et elle peut être : `public` (`+`) quand toutes les autres classes ont accès, `protected` (`#`) quand seules la classe elle-même et les classes filles (héritage) ont accès et `private` (`-`) quand seule la classe elle-même a accès.

Systèmes d'exploitation

Android
Android est un système d'exploitation mobile, basé sur le noyau Linux et développé actuellement par Google. Le système avait d'abord été conçu pour les *smartphones* et tablettes tactiles, puis s'est diversifié dans les objets connectés et ordinateurs comme les télévisions (Android TV), les

voitures (Android Auto), les ordinateurs (Android-x86) et les *smartwatch* (Android Wear). En 2015, Android est le système d'exploitation mobile le plus utilisé dans le monde avec plus de 80 % de parts de marché dans les *smartphones*, devant iOS d'Apple. Le développement d'applications pour Android s'effectue avec un ordinateur personnel sous Mac OS, Windows ou Linux en utilisant le *Java Development Kit* (JDK) de la plate-forme Java et des outils pour Android.

GNU
: *GNU's Not UNIX* (GNU) est un système d'exploitation libre créé en 1983 par Richard Stallman, maintenu par le projet GNU. Il reprend les concepts et le fonctionnement d'Unix. Le projet GNU possède une version utilisable de tous les éléments nécessaires à la construction d'un système d'exploitation (un *shell*, des bibliothèques, les composants de base, les outils de développement, ...) et d'un noyau (*kernel*) nommé Hurd. Les logiciels qui composent GNU sont généralement utilisés en association avec des logiciels libres issus d'autres projets tels que le noyau Linux. GNU/Linux (souvent appelé Linux) est une variante du système d'exploitation GNU fonctionnant avec le noyau Linux.

iOS
: iOS, anciennement iPhone OS, est le système d'exploitation mobile développé par Apple pour plusieurs de ses appareils. Il est dérivé de Mac OS dont il partage les fondations.

Linux
: Linux est le nom couramment donné à tout système d'exploitation libre fonctionnant avec le noyau Linux. L'appellation exacte est **GNU/Linux**.

Mac OS
: Mac OS est un système d'exploitation à interface graphique développé par Apple pour équiper ses ordinateurs personnels Macintosh, de 1984 à 2001. C'est le premier système grand public ayant une interface graphique, inspirée de Xerox Alto et fondée sur le modèle WIMP (c.-à-d. doté de fenêtres, icônes, menus et souris).

Multi-tâche
: Un système d'exploitation est dit multi-tâches quand il permet l'exécution simultanée de plusieurs programmes. Tous les systèmes d'exploitation actuels sont multi-tâches.

Multi-utilisateur
: Un système d'exploitation est dit multi-utilisateurs quand il est conçu pour être utilisé simultanément par plusieurs usagers, souvent à travers un réseau informatique (notion de serveurs). Ils sont multi-tâches et en général sécurisés, c'est-à-dire qu'il vont refuser d'exécuter toute opération pour laquelle l'usager n'a pas préalablement reçu une permission.

Système d'exploitation
: Un Système d'Exploitation (SE) ou *Operating System* (OS) est un ensemble de programmes d'un équipement informatique qui sert d'interface entre le matériel et les logiciels applicatifs. C'est donc une couche logicielle (*software*) qui permet et coordonne l'utilisation du matériel (*hardware*) entre les différents programmes d'application. Un système d'exploitation est typiquement composé : d'un noyau (*kernel*), de bibliothèques, d'un ensemble d'outils système et de programmes applicatifs de base. Tous les systèmes d'exploitation actuels sont multi-tâches et multi-utilisateurs.

Unix
: Unix, officiellement UNIX, est un système d'exploitation multitâche et multi-utilisateur créé en 1969 par Kenneth Thompson. Il repose sur un interpréteur (le *shell*) et de nombreux petits utilitaires, accomplissant chacun une action spécifique, commutables entre eux (mécanisme de « redirection ») et appelés depuis la ligne de commande. Il a donné naissance à une famille de systèmes, dont les plus populaires à ce jour sont les variantes de BSD (notamment FreeBSD, NetBSD et OpenBSD), GNU/Linux, iOS et Mac OS.

Windows
Windows est au départ une interface graphique unifiée produite par Microsoft, qui est devenue ensuite une gamme de systèmes d'exploitation à part entière, principalement destinés aux ordinateurs compatibles PC.

Base de données

attribut
Un attribut est une caractéristique d'une table susceptible d'être enregistrée dans la base de données. Par exemple une personne (table), son nom et son adresse (des attributs). Les attributs sont également appelés des champs ou des colonnes.

base de données
Une base de données est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour et recherche de données).

base de données relationnelle
C'est une base de données organisée selon un modèle de données de type relationnel, à l'aide d'un Système de Gestion de Bases de Données (SGBD). Une base de données relationnelle est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des relations ou **tables**. Les lignes de ces relations sont appelées des *nuplets* (tuples) ou **enregistrements**. Les noms des **colonnes** (ou champs) sont appelées des **attributs**.

cardinalité
La cardinalité d'un lien entre deux tables A et B par exemple est le nombre de A pour lesquelles il existe un B et inversement. Celle-ci peut être un-a-un ou 1:1 (un enregistrement de la table A se rapporte seulement à un enregistrement de la table B), un-a-plusieurs ou 1:N (un enregistrement de la table A se rapporte à un ou plusieurs enregistrements de la table B), et plusieurs-à-plusieurs ou N:N (un enregistrement de la table A se rapporte à un ou plusieurs enregistrements de la table B et un enregistrement de la table B se rapporte à un ou plusieurs enregistrements de la table A). Une relation N:N peut donc être décomposées en deux relations 1:N.

clé étrangère
Dans les modèles de données relationnels, une clé étrangère identifie une colonne (ou un ensemble de colonnes) d'une table comme référençant une colonne (ou un ensemble de colonnes) d'une autre table. Les colonnes de la table référencée doivent faire partie d'une contrainte de clé primaire ou d'une contrainte d'unicité. Une contrainte de clé étrangère permet ainsi d'établir des liens entre plusieurs tables : il s'agit d'un des principes fondamentaux des bases de données relationnelles.

clé primaire
Dans les modèles de données relationnels, la clé primaire est un attribut dont le contenu est différent pour chaque enregistrement de la table, ce qui permet de retrouver un et un seul enregistrement (unicité).

enregistrement
Un enregistrement est une donnée qui comporte plusieurs champs dans chacun desquels est enregistré une donnée.

modèle de données
Le schéma ou modèle de données est la description de l'organisation des données. Il renseigne sur les caractéristiques de chaque type de donnée et les relations entre les différentes données qui se trouvent dans la base de données. Il existe plusieurs types de modèles de données (relationnel, entité-association, objet, hiérarchique et réseau).

modèle de données relationnel
C'est le type de modèle de données le plus couramment utilisé pour la réalisation d'une base de données. Selon ce type de modèle, la base de données est composée d'un ensemble de tables dans lesquelles sont placées les données ainsi que les liens. Chaque ligne d'une table est un enregistrement.

MySQL
MySQL est un Système de Gestion de Bases de Données Relationnelles (SGBDR). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et Microsoft SQL Server. MySQL est un serveur de bases de données relationnelles *Structured Query Language* (SQL). Il est multi-thread et multi-utilisateur. MySQL fonctionne sur de nombreux systèmes d'exploitation différents, incluant Linux, Mac OS et Windows. Les bases de données sont accessibles en utilisant de très nombreux langages de programmation C, C++, C#, Java, Perl, PHP, Python, ... Une API spécifique est disponible pour chacun d'entre eux. MySQL fait partie du quatuor LAMP : Linux, Apache, MySQL, PHP. Il appartient également à ses variantes WAMP (Windows) et MAMP (Mac OS). En 2009, à la suite du rachat de MySQL, Michael Widenius, fondateur de MySQL, lance le projet MariaDB, dans une démarche visant à remplacer MySQL.

SQL
C'est un langage normalisé de requête structurée servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher (SELECT), d'ajouter (INSERT), de modifier (UPDATE) ou de supprimer (DELETE) des données dans les bases de données relationnelles.

SQLite
SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL. Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être un fichier directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme. SQLite est le moteur de base de données le plus distribué au monde, grâce à son utilisation dans de nombreux logiciels grand public comme Firefox, Skype, Google Gears, dans certains produits d'Apple, d'Adobe et de McAfee et dans les bibliothèques standards de nombreux langages comme PHP ou Python. De par son extrême légèreté (moins de 300 Kio), il est également très populaire sur les systèmes embarqués, notamment sur la plupart des smartphones modernes : l'iPhone ainsi que les systèmes d' exploitation mobiles Symbian et Android l'utilisent comme base de données embarquée..

système de gestion de bases de données relationnelles
Les logiciels qui permettent de créer, utiliser et maintenir des bases de données relationnelles sont des systèmes de gestion de bases de données relationnelles (SGBDR).

UML

acteur
Un acteur (*actor*) représente un rôle joué par une entité externe qui interagit avec le système. Un acteur est identifé par un nom et peut être : un humain, un dispositif matériel ou un autre système.

action
Une action est le plus petit traitement qui puisse être exprimé en *Unified Modeling Language* (UML) : c'est une opération atomique ininterruptible. Une action a une incidence sur l'état du système ou en extrait une information.

activité
: Une activité définit un comportement décrit par un séquencement organisé d'actions. Le flot d'exécution est modélisé par des noeuds reliés par des transitions. Le flot de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés.

agrégation
: Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance. Les agrégations n'ont pas besoin d'être nommées : implicitement elles signifient « contient » ou « est composé de ». Elle est représentée par trait plein avec ou sans flèche et un losange vide côté composite. Dans une agrégation, le composant peut être partagé entre plusieurs composites ce qui entraîne que, lorsque le composite sera détruit, le composant ne le sera pas forcément. La relation d'agrégation est une relation sémantique de type « avoir ».

artefact
: Un artefact est une manière de définir un élément concret (fichier, un programme, une bibliothèque ou une base de données) construit ou modifié dans un projet.

association
: Une association représente une relation sémantique durable entre deux classes. Les associations peuvent donc être nommées pour donner un sens précis à la relation. Elle est représentée par trait plein avec ou sans flèche. La relation d'association est une relation sémantique de type « avoir ».

cas d'utilisation
: Un cas d'utilisation (*use case*) représente une fonction offerte par le système et qui produit un résultat observable intéressant pour un acteur. Chaque cas d'utilisation spécifie un comportement attendu du système. Il permet de décrire ce que le système devra faire, sans spécifier comment il le fera. On nommera les cas d'utilisation par un verbe à l'infinitif suivi d'un complément, du point de vue de l'acteur (et non du système).

classe abstraite
: Une classe est dite abstraite lorsqu'on ne peut instancier d'objets à partir de cette classe. Le nom des classes abstraites est écrit en *italique*. On peut ajouter le stéréotype `«abstract»`.

classe active
: Une classe active est une classe qui possède une méthode qui s'exécute dans un flot de contrôle distinct. Ce flot est généralement le fil d'exécution d'un *thread*. Une instance d'une classe active sera nommée object actif. UML fournit un repère visuel (bord en trait épais ou double trait) qui permet de distinguer les éléments actifs des éléments passifs. Il est tout de même conseillé d'ajouter le stéréotype `«thread»`.

composition
: Une composition est une agrégation plus forte signifiant « est composée d'un » et impliquant : qu'une partie ne peut appartenir qu'à un seul composite (agrégation non partagée) et que la destruction du composite entraîne la destruction de toutes ses parties (il est responsable du cycle de vie de ses parties). Elle est représentée par trait plein avec ou sans flèche et un losange plein côté composite. La relation de composition est une relation sémantique de type « avoir ».

dépendance
: Une dépendance s'illustre par une flèche en pointillée en UML. Elle indique qu'un élément « a besoin » des services d'un autre élément. Lorsqu'un objet « utilise » temporairement les services d'un autre objet, cette relation d'utilisation est une dépendance entre classes. La plupart du temps, les dépendances servent à montrer qu'une classe utilise une autre comme argument dans la signature d'une méthode. On parle aussi de lien temporaire car il ne dure que le temps de l'exécution de la méthode. Cela peut être aussi le cas d'un objet local à une méthode. Généralement, les dépendances ne sont pas montrées dans un diagramme de classes car elles ne sont qu'une utilisation temporaire donc un détail de l'implémentation que l'on ne considère pas judicieux de mettre en avant.

diagramme

Les diagrammes sont des ensembles d'éléments graphiques (pictogrammes). Ils décrivent le contenu des vues et peuvent faire partie de plusieurs vues. Il en existe quatorze depuis UML 2.3. UML n'étant pas une méthode de développement, leur utilisation est laissée à l'appréciation de chacun. Des méthodologies, telles que l'*Unified Process* (UP), axent l'analyse en tout premier lieu sur les diagrammes de cas d'utilisation (*use case*).

diagramme d'activité

Un diagramme d'activité (*activity diagram*) est une représentation du comportement du système ou de ses composants sous forme de flux ou d'enchaînement d'activités. Il décrit les activités séquentielles et parallèles d'un système. Ils permettent ainsi de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.

diagramme de classes

Le diagramme de classes (*class diagram*) représente les classes intervenant dans le système et leurs relations.

diagramme de composants

Le diagramme de composants (*component diagram*) décrit l'organisation du système du point de vue des éléments logiciels comme les modules (paquets, fichiers sources, bibliothèques, exécutables), des données (fichiers, bases de données) ou encore d'éléments de configuration (paramètres, scripts, fichiers de commandes). Ce diagramme permet notamment de mettre en évidence les dépendances entre les composants (qui utilise quoi).

diagramme de déploiement

Le diagramme de déploiement (*deployment diagram*) est une représentation des éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage, ...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux. Il présente le déploiement des éléments logiciels sur l'architecture physique. Les caractéristiques des ressources matérielles physiques et des supports de communication peuvent être précisées par stéréotype (`«USB»`, `«RS485»`, ...).

diagramme de séquence

Un diagramme de séquence (*sequence diagram*) est un diagramme d'interaction dont le but est de décrire comment les objets collaborent au cours du temps et quelles responsabilités ils assument. Il décrit un scénario d'un cas d'utilisation. Un diagramme de séquence représente donc les interactions entre les éléments du système et/ou de ses acteurs, en insistant sur la chronologie des envois de message. C'est un diagramme qui représente la structure dynamique d'un système car il utilise une représentation temporelle. Les objets, intervenant dans l'interaction, sont matérialisés par une « ligne de vie », et les messages échangés au cours du temps sont mentionnés sous une forme textuelle.

diagramme de séquence système

Un Diagramme de Séquence Système (DSS) permet de décrire le comportement du système vu de l'extérieur (par les acteurs) sans préjuger de comment il sera réalisé. Le système est vu comme une « boîte noire » qui sera ouverte (décrite) seulement plus tard en conception..

diagramme des cas d'utilisation

Le diagramme des cas d'utilisation (*use-case diagram*) permet une représentation des possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire de toutes les fonctionnalités que doit fournir le système.

diagramme états-transitions

Le diagramme d'états ou états-transitions (*state machine diagram*) représente le comportement du système ou de ses composants sous forme de machine à états finis. Il est souvent utilisé pour décrire le comportement interne d'un objet en représentant les séquences possibles d'états et d'actions. Il peut être aussi utiliser pour spécifier le comportement interne d'autres éléments tels que les cas d'utilisation, les sous-systèmes, les méthodes.

élément

Les modèles d'élément sont les éléments graphiques (pictogrammes) des diagrammes.

extension (use case)
Dans une relation d'extension (`«extend»`), le cas d'utilisation de base incorpore implicitement un autre cas de façon optionnelle.

fragment
Un fragment (ou cadre) permet d'identifier une sous-partie d'une interaction afin que celle-ci soit référencées par d'autres interactions ou de lui spécifier des conditions particulières d'exécution (boucle, optionnel, ...). Il existe par exemple les fragments suivants : `sd` (fragment du diagramme de séquence en entier), `alt` (fragment alternatif Si ... Alors ... Sinon ...), `opt` (fragment optionnel), `par` (fragment parallèle pour les traitements concurrents), `loop` (le fragment s'exécute plusieurs fois dans une boucle), `region` (région critique où un seul thread peut s'exécuter à la fois), `ref` (référence à une interaction dans un autre diagramme) ....

généralisation (use case)
Dans une relation de généralisation/spécialisation (héritage), les cas d'utilisation descendants héritent de leur parent commun. Il est également possible d'appliquer à un acteur la relation de généralisation. Cela se fait notamment lorsqu'un acteur est un sous-type d'une autre catégorie d'acteurs. Un acteur lié à un autre par un ce type de relation peut interagir avec le système de plus de manières que son parent.

héritage
L'héritage est un concept fondamental de la programmation orientée objet. Il se nomme ainsi car le principe est en quelque sorte le même que celui d'un arbre généalogique. Ce principe est fondé sur des classes « filles » qui héritent des caractéristiques des classses « mères ». L'héritage permet d'ajouter des propriétés à une classe existante pour en obtenir une nouvelle plus précise. Il permet donc la spécialisation ou la dérivation de types. L'héritage est représenté par un trait reliant les deux classes et dont l'origine (classe mère) se distingue de l'autre extrémité (classe fille) par un triangle vide. La relation d'héritage est une relation sémantique de type « être ».

inclusion (use case)
Dans une relation d'inclusion (`«include»`), le cas d'utilisation de base incorpore explicitement un autre cas de façon obligatoire.

message
Les objets interagissent entre eux en s'échangeant des messages. La réponse à la réception d'un message par un objet est appelée une méthode. Une méthode est donc la mise en oeuvre du message : elle décrit la réponse qui doit être donnée au message. Dans un diagramme de séquence, une activité représente l'exécution d'une méthode. On distingue deux types de message : synchrone où l'objet émetteur se bloque en attendant la réponse de l'objet récepteur du message et asynchrone où l'objet émetteur n'attend pas la réponse de l'objet récepteur du message et continue son activité.

modélisation
La modélisation consiste à créer une représentation simplifiée d'un problème : le modèle. Le modèle constitue ainsi une représentation possible du système pour un point de vue donné. La modélisation comporte deux composantes : l'analyse, c'est-à-dire l'étude du problème et des besoins (quoi faire ?) et la conception, qui l'élaboration d'une solution au problème et aux besoins (comment faire ?).

multiplicité
Aux extrémités d'une association, agrégation ou composition, il est possible d'y indiquer une multiplicité (ou cardinalité) : c'est pour préciser le nombre d'instances (objets) qui participent à la relation. Une multiplicité peut s'écrire : n (exactement n, un entier positif), n..m (n à m), n..* (n ou plus) ou * (plusieurs).

navigabilité
Aux extrémités d'une association, agrégation ou composition, il est possible d'ajouter une flèche

sur la relation ce qui précise la navigabilité. La navigabilité permet de préciser « qui connaît qui » dans la relation. Les relations peuvent être bidirectionnelles (pas de flèche) ou unidirectionnelle (avec une flèche qui précise le sens).

paquet
Un paquet (*package*) est un mécanisme général de regroupement d'éléments (diagrammes, cas d'utilisation, acteurs, classes, ...).

relation (classes)
Étant donné que les objets logiciels interagissent entre eux, il existe donc des relations entre les classes. On distingue cinq différents types de relations de base entre les classes : l'association (trait plein avec ou sans flèche), la composition (trait plein avec ou sans flèche et un losange plein), l'agrégation (trait plein avec ou sans flèche et un losange vide), la relation de généralisation ou d'héritage (flèche fermée vide) et la dépendance (flèche pointillée). Il existe des relations durables (association, composition, agrégation, héritage) et des relations temporaires (dépendance).

relation (use case)
Pour affiner les diagrammes, il est possible d'ajouter des relations entre cas d'utilisation. UML définit trois types de relations standardisées : inclusion, extension et généralisation.

rôle
À l'extrémité d'une association, agrégation ou composition, on donne un nom : c'est le rôle de la relation. Par extension, c'est la manière dont les instances d'une classe voient les instances d'une autre classe au travers de la relation.

stéréotype
Un stéréotype est une marque de généralisation notée par des guillemets (« »), cela montre que l'objet est une variété d'un modèle.

transition
Une transition est le passage d'un état dans un autre si un événement déclencheur se produit (et que la condition de garde est vérifiée). Cela provoque l'exécution de certaines activités.

uml
UML est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet. UML est utilisé pour spécifier, visualiser, modifier et construire les documents nécessaires au bon développement d'un logiciel orienté objet. UML permet d'obtenir une modélisation indépendante des langages et des environnements.

visibilité
La notion de visibilité indique qui peut avoir accès à l'élément. La visibilité est précisée au sein des classes (et non des objets) et elle peut être : `public` (`+`) quand toutes les autres classes ont accès, `protected` (`#`) quand seules la classe elle-même et les classes filles (héritage) ont accès, `private` (`-`) quand seule la classe elle-même a accès et `package` (`~`) quand la classe est visible uniquement dans le paquet.

vue
Les vues décrivent le système d'un point de vue donné, qui peut être organisationnel, dynamique, temporel, architectural, géographique, logique, etc. En combinant toutes ces vues, il est possible de définir (ou retrouver) le système complet. Les vues sont décrites par des diagrammes.

vue des cas d'utilisation
La vue des cas d'utilisation (*use-case view*) est la description du modèle vu par les acteurs du système. Elle correspond aux besoins attendus par chaque acteur.