heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

comem+

Programmation orientée objet

Support de cours

Table des matières

Chapitre 1

Introduction

1.1 La programmation objet

Contrairement à la programmation procédurale où l'on utilise une approche de recherche de la solution du "haut vers le bas" en analysant le problème dans sa globalité et en le découpant ensuite en sous problèmes (qui deviendront lors de l'étape de la programmation des procédures), et en recherchant les données en entrées et en sorties du problème et de chacun des sous problèmes, la programmation orientée objet aborde une méthodologie différente.

Lors de l'analyse du problème, la première phase consiste à identifier les différents types d'entités (concrètes ou abstraites) qui le composent. Chaque type d'entité (nommé *classe*) sera ensuite décrit selon ses données et son comportement. Une des difficultés sera de savoir quel comportement un objet doit avoir et surtout quels messages il devra pouvoir interpréter ou fournir au monde extérieur.

Prenons en exemple un problème simple : calculer et afficher la circonférence et la surface d'un ensemble de cercles. On identifie très facilement "Cercle" comme étant une classe, mais quand est-il de "surface" et de "circonférence" ? Sont-ils des classes, des données d'un cercle, les deux à la fois, ou sont-ils simplement des messages (des informations) qu'un cercle doit fournir ? La réponse va influencer fortement l'étape de programmation, il faut donc choisir judicieusement. Considérons les comme des informations que le cercle doit pouvoir fournir au monde extérieur. Alors quelles sont les données qui composent un cercle ? Si on réfléchi bien au problème, la seul donnée importante qui permet de résoudre le problème est le rayon. Dans la terminologie de la programmation objet on nomme ses données par les termes *propriétés* ou *attributs*.

Maintenant, intéressons nous aux messages ou actions qu'un cercle doit pouvoir fournir. En relisant l'énoncé du problème on en déduit qu'un cercle doit pouvoir calculer sa circonférence et aussi calculer sa surface. On appelle ces actions des *méthodes* de la classe "Cercle". Qu'en est-il de l'affichage de ces informations ? Est-ce au cercle de s'occuper de cela ? Non, car le cercle ne connaît en aucune manière la façon dont nous voulons afficher les résultats. Nous avons donc oublié une entité durant notre première analyse, en effet le problème ne spécifie rien sur la manière d'afficher les résultats, c'est donc à nous de choisir. Pour simplifier, prenons la sortie standard, c'est à dire l'écran.

Au final, nous avons deux classes : "Cercle" et "Écran". Un cercle à un seul *attribut*, le rayon, et deux *méthodes*, calculer la surface et calculer la circonférence. Pour ce qui est de la classe "Écran", il est difficile de lui donner des attributs, mais il doit y avoir au moins une méthode d'affichage d'un message sur l'écran. Heureusement pour nous, nous verrons que la classe gérant l'écran existe déjà dans la plupart des langages de programmation orienté objet(y compris Java).

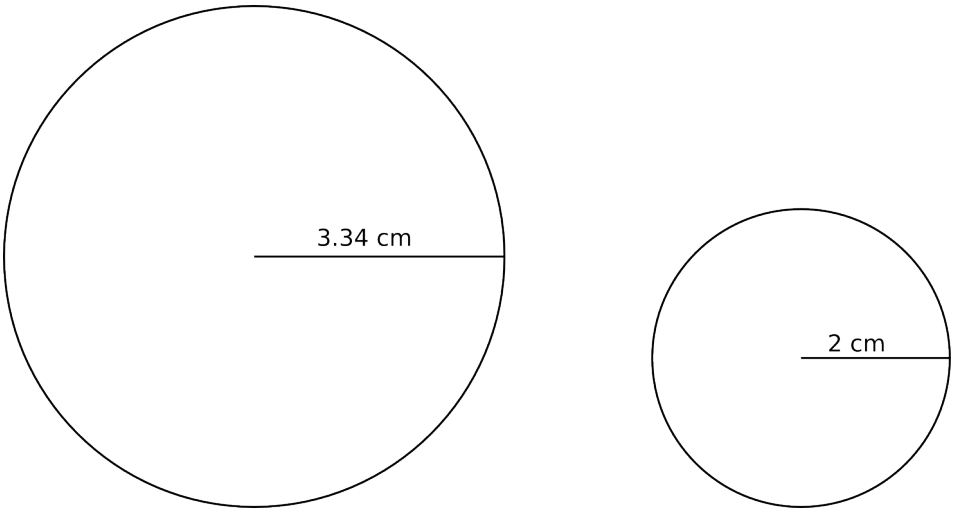

FIGURE 1.1 – Deux cercles

1.2 Classes, instances et constructeurs

Les classes représentent donc les types d'entités qui composent notre problème. Elles permettent de définir les attributs (ou propriétés) ainsi que les méthodes (fonctionnalités, actions, ...) qui composent ces entités. Mais qu'en est-il des entités elles mêmes ? Qu'en est-il des deux cercles de la figure 1.1 ? Les objets (comme ces deux cercles par exemple) sont construit à partir d'une classe. Ce mécanisme s'appelle *l'instanciation*, autrement dis la construction d'une *instance* (ou objet) se fait obligatoirement à partir d'une classe. On ne peut donc crée des instances "cercle" que si la classe "Cercle" existe. Il est très important de comprendre la différence entre objet et classe, la classe représente le type d'entité et permet de définir ses propriétés et son comportement (par exemple : un cercle a un rayon, il doit être capable de donner sa surface et sa circonférence, ...), tandis qu'un objet représente une entité qui possède des données propres à elle même (un rayon dans le cas de nos cercles). Pour faire le parallèle avec les types de base, nous pouvons rapprocher une classe à une type et un objet à une variable.

La création d'une instance (l'instanciation) se fait d'une manière bien précise. Premièrement, dans la classe une méthode permettant de construire les objets doit exister. Cette méthode particulière est appelée le *constructeur*, elle est différente des autres méthodes par le fait que ce n'est pas une fonctionnalité d'un objet mais bien une fonctionnalité de la classe elle même, la classe se charge donc de la création des instances. Pour la différencier des autres méthodes, les langages de programmation objet utilisent une syntaxe particulière pour la définir. Normalement toutes classes doit avoir un constructeur, sinon aucune création d'objet n'est possible, mais il existe des exceptions. De plus, une classe pourrait avoir plusieurs constructeurs, chacun définissant de manière différente comment créer une instance. Laissons ces cas particuliers pour plus tard pour nous intéresser au rôle concret du constructeur. Pour les cas simples, le constructeur doit juste renseigner les attributs de l'objet en création. C'est à dire qu'il va recevoir en paramètre les données permettant de renseigner tous les attributs nécessaires à la création d'une instance. Dans le cas de nos cercles, comme le seul attribut est le rayon, le constructeur de la classe "Cercle" est donc très simple : il reçoit en paramètre une valeur représentant le rayon et sauve cette valeur dans l'attribut "rayon" de la nouvelle instance.

1.3 Encapsulation des attributs

Les attributs d'une instance sont normalement inaccessibles au monde extérieur. Un objet ne doit pas donner libre accès à ses données et ceci pour plusieurs bonnes raisons. Premièrement ces données lui appartiennent, il lui incombe donc aussi de les modifier. Cela serait problématique si, sans qu'on

l'avertisse, un autre objet modifie son rayon. Ou pire, imaginer une donnée sensible comme le mot de passe d'un utilisateur, il serait fort regrettable que celui-ci soit visible pat tous les autres objets. De même, si un cercle une fois créer ne doit pas pouvoir changer son rayon, il serait souhaitable que personne ne puisse le faire. Une deuxième raison est l'évolution du code. Si personne ne peut accéder aux données propre à un objet, celui-ci peut modifier sa structure interne sans que le monde extérieur ne remarque quoique ce soit. Par exemple pour le cercle de la figure 1.2, si on change le nom de l'attribut "rayon" mais que la méthode "getRayon" continu d'exister personne ne verra la différence. Il y a beaucoup d'autres exemples qui montre que l'encapsulation est non seulement utile, mais fortement conseillée pour une programmation objet propre et efficace.

Nous pouvons nous représenter un objet sous la forme d'une bouée (cf. figure 1.2), les méthodes se situent sur le pourtour tandis que les données sont bien à l'abri à l'intérieur. Seul les méthodes de l'objet peuvent interagir avec les données internes en respectant ainsi le principe d'encapsulation des attributs. Si nous voulons rendre accessible une donnée interne, il nous faut alors créer une méthode permettant de fournir cette information. Par exemple nous souhaitons que le cercle puissent communiquer la valeur de son rayon, pour se faire nous créons une méthode toute simple qui ne fera que retourner la valeur du rayon. Cette méthode particulière est appelée *accesseur*. Si nous voulons que l'on puisse aussi modifier le rayon d'un cercle, nous pouvons créer une méthode qui recevra en paramètre la nouvelle valeur du rayon et l'affectera à l'attribut correspondant, il s'agit là d'une méthode appelée *modificateur*. Les accesseurs et les modificateurs doivent ou ne doivent pas exister selon que l'on veuille donner accès à la lecture ou la modification des attributs qui composent l'objet.

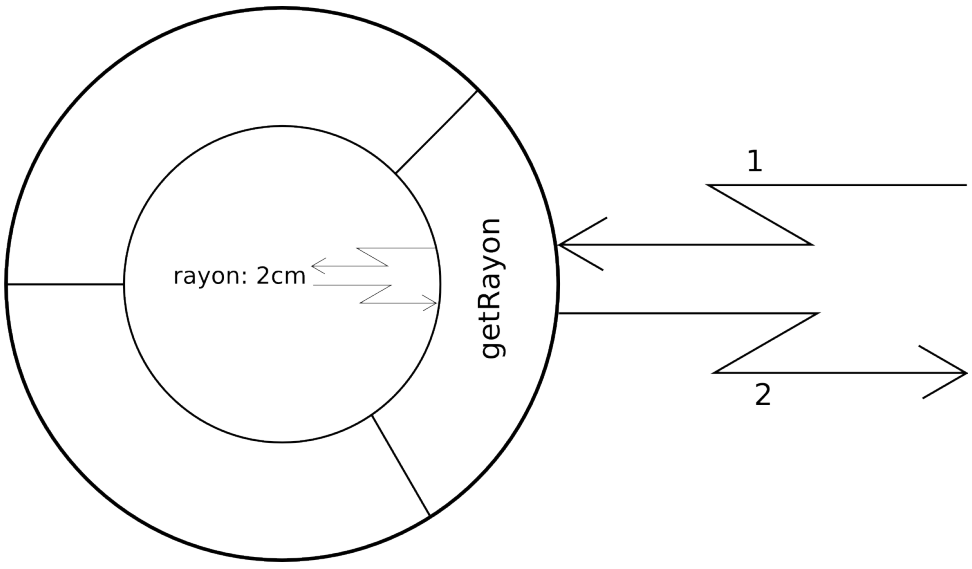

Figure 1.2 – Un cercle