Programmation Orienté Objet

Mendy Fatnassi

10 décembre 2020

Table des matières

Chapitre 1 : POO

1.1 Langage Orienté Objet

Un Langage Orienté Objet (LOO) $\neq$ Langage Objet (LO).
Il existe plusieur langage utilisant les principe de la POO tel que :
Ruby,JAVA,C++,SmallTalk,Eiffel,Ada,Python,...

1.2 Notion et Concept de base

1.2.1 Objet

:

Identité : Signifie que les données sont regroupés en entités discrètes et identifiables appelés objets.

Dans le monde réel nous somme entouré d'objet tel que une chaise, une voiture, une table ,etc. Ces objets posséde :
-Des propriété, la chaise possède 4 pieds, elle est de couleur bleue, etc.
-Peuvent faire des actions, la voiture peut rouler, klaxonner, etc.
-Ils peuvent également interagir entre eux (l'objet conducteur démarre la voiture, l'objet voiture fait tourner l'objet volant, etc.).
Sachant qu'un objet en programmation c'est comme un objet du monde réel mais ce n'est pas forcément restreint au matériel. Un chien est un objet. Des concepts comme l'amour ou une idée sont également des objets, tandis qu'on ne dirait pas cela dans le monde réel.
Un objet est responsable des valeurs des variables d'etat (attributs).

1.2.2 Instance

:

Il faut bien faire attention à distinguer ce qu'est l'objet et ce qu'est la définition d'un objet.

La définition de l'objet (ou structure de l'objet) permet d'indiquer ce qui compose un objet, c'est-à-dire quelles sont ses propriétés, ses actions etc. Comme par exemple le fait qu'une chaise ait des pieds ou qu'on puisse s'asseoir dessus.
Par contre, l'objet chaise est bien concret. On peut donc avoir plusieurs objets chaises : on parle également d'**instances**.

1.2.3 Class

:

Un objet (concret) peut etre definit par une classe (objet abstrait).
Une instance d'une class est un objet.

Par exemple si on prend ce texte :
"un compte en banque *connait* un numero, un titulaire et un solde et *est capable* de donner le solde, de deposer une somme et de retirer une somme "

-Ce texte definit monCompte,tonCompte,sonCompte,unAutreCompte etc..
-Et definit la class Compte a partir de laquelle on pourras creer tous les autres comptes particulier.

Une classe est donc structuré de la facons suivante :
-Une liste de noms de champs (variables d'instances).
-Un catalogue de procédures (méthodes).

Methode de class :
A ne pas confondre avec une methode d'instance qui agit que sur un seul objet (instance de classe) a la fois.
Une methode de class (statique) est indépendante de toute instance de la classe.

1.2.4 Methode

:
Il s'agit du comportement d'un objet decrit par un ensemble de procédure (opération).
Appelé fonction membres (en C++), méthode (en ruby.

1.2.5 Interface

On ne peut communiquer avec un objet que par son interfaces (les services qu'il offrent).

1.3 Concepte fondamentaux de la POO

:

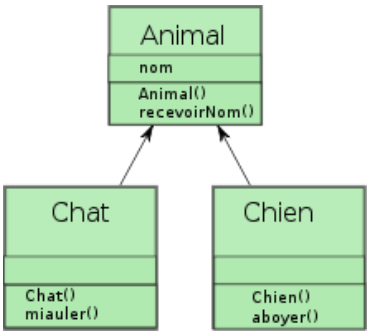
Variable d'etat d'un objet : Il s'agit de champs qui serve a decrire la class.
Appelé données membres (en C++), **Variable d'instance** (en Ruby).
Message : Un objet (emetteur) envoie un message a un objet (receveur). Un message permet d'activer une méthode de l'objet receveur.
En reponse a un message l'objet destinataire du message déclenche un comportement.

1.3.1 Heritage

:

C'est-à-dire qu'un objet dit "père" peut transmettre certaines de ses caractéristiques à un autre objet dit "fils".
Pour cela, on pourra définir une relation d'héritage entre eux. S'il y a une relation d'héritage entre un objet père et un objet fils, alors l'objet fils hérite de l'objet père. On dit également que l'objet fils est une **spécialisation** de l'objet père ou qu'il dérive de l'objet père.
A l'inverse on peux adopter une méthode de **généralisation** de l'objet père càd on commence par definir l'objet fils et ensuite on definit l'objet père (le fils heritera tout pareil du père).

Grace a cela nous pouvons etablir une hiéarchie des classes. exemple :



1.3.2 Classification

:

Les objets ayant la meme structure de données et les memes services sont regroupés dans une classe.

1.3.3 Abstraction

:

L'abstraction est une vue générique d'une entité permettant de ne pas etre limité par des contraintes technique ou matériel.
Par exemple la structure personne peut definir une infinité d'objet qui seront isomorphe (meme structure), on auras donc plusieurs personne (pers1,pers2,...). Leurs comportement seras identique car ils partagent l'ensemble des procédures.
Le texte definissant un objet (concret) devient une classe (objet abstrait) définissnt une infinité d'instances.

1.3.4 Polymorphisme

:

Le mot polymorphisme suggère qu'une chose peut prendre plusieurs formes. Sous ce terme un peu barbare se cachent plusieurs notions de l'orienté objet qui sont souvent sources d'erreurs.

C'est la capacité pour un objet de faire une même action avec différents

types d'intervenants. C'est ce qu'on appelle le polymorphisme "ad hoc" ou le polymorphisme "paramétré".

Par exemple, notre objet voiture peut rouler sur la route, rouler sur l'autoroute, rouler sur la terre si elle est équipée de pneus adéquats, rouler au fond de l'eau si elle est amphibie, etc.
Concrètement ici, je fais interagir un objet "voiture" avec un objet "autoroute" ou un objet "terre", par l'action qui consiste à "rouler".

La substitution est une autre manifestation du polymorphisme. Il s'agit de la capacité d'un objet fils à redéfinir des caractéristiques ou des actions d'un objet père.

Prenons par exemple un objet mammifère qui sait faire l'action "se déplacer". Les objets qui dérivent du mammifère peuvent potentiellement avoir à se déplacer d'une manière différente. Par exemple, l'objet homme va se déplacer sur ses deux jambes et donc différemment de l'objet dauphin qui se déplacera grâce à ses nageoires, etc.

Tous ces mammifères sont capables de se déplacer, mais chacun va le faire d'une manière différente. Ceci est donc possible grâce à la substitution qui permet de redéfinir un comportement hérité. Ainsi, chaque fils sera libre de réécrire son propre comportement, si celui de son père ne lui convient pas.

1.3.5 Encapsulation

En POO un objet a des donnée internes. Ces données sont spécifié dans la classe.
En ruby l'unité d'encapsulation est la classe.
Un objet est responsable de son fonctionnement, de ces méthodes, il ne se laisse pas manipuler par un autre objet.
Les objet vont plutot s'envoyer et manipuler des messages.

L'écriture de classes offre d'autres avantages que le simple regroupement de données et de traitements. Parmi ceux-ci figure la possibilité de restreindre l'accès à certains éléments de la classe. C'est ce que l'on appelle l'encapsulation.
Exemple :

```
class CompteBancaire
{
    private titulaire;   // attribut privé
    public solde;
    private devise;      // attribut privé
    ...
    du code
    ...
}
```

À présent, la seule manière de définir des valeurs pour titulaire et devise est d'utiliser le constructeur. Toute tentative d'accès externe aux propriétés privées générera une erreur lors de la compilation.

```
comptePierre = new("Pierre", 0, "euros");

comptePierre.titulaire = "Pierre";  // Erreur : titulaire est un attribut pri
comptePierre.solde = 500;           // OK : solde est un attribut public
comptePierre.devise = "euros";      // Erreur : devise est un attribut privé
```

Niveau d'encapsulation :

- **public** : Les méthodes de la classe peuvent etre appelé depuis n'importe ou dans le code (classe par défaut public).
- **private** : La méthode ne pourra être appelée qu'à l'intérieur du code de la classe, sans indiquer de récepteur.
- protected : la méthode ne pourra être appelée qu'à l'intérieur du code de la classe ou de une de ses classes filles.

Encapsulation des attributs :

En Ruby, tout attribut d'instance est par définition privé. Pour déclarer que l'on peut y accéder en lecture il faut définir une méthode d'accès en lecture grâce à la syntaxe : attr_reader,attr_writer ou attr_accessor .

1.4 Constructeur/Accesseur

Constructeur : methode initialize et new.
chaque appele de la methode new ferass automatiquement appele a initialize . Si initialize(param) a des parametre , alors la méthode new doit aussi avoir des paramétres.

Un accesseur est une méthode le plus souvent publique qui permet d'accéder à un attribut privé.

Un accesseur en lecture (getter) permet de lire la valeur d'un attribut.
Un accesseur en écriture (mutateur ou setter) permet de modifier la valeur d'un attribut.

```
attr_reader :symbol[,:symbol]
attr_writer :symbol[,:symbol]
```

On peux definir un raccourcie qui reunie les attribut en lecture et ecriture : attr_accessor

Chapitre 2 : Ruby

2.1 Généralité

Ruby est un lagage open-source dynamique $\neq$ statique. Le C est un lagage statique ou typé alors qu'avec ruby les objet n'on pas besoins d'etre spécifié par un type de donnée.

Pour lancer un programme ruby ils nous suffit de taper dans un shell : *$ruby prog.rb* ou alors on peux lancer l'interpreteur ruby **irb**.

On peux rendre un programme ruby executable comme tout autre programme. Pour commencer on doit savoir ou se trouve le dossier de ruby :

```
$which ruby
usr/local/bin/ruby
```

Ensuite on recopie le chemin dans la toute premiere ligne de notre script ruby avec le prefixe "!/path/".

Installation : Ruby,MSYS2,Glade,Gtk.

Ruby est extensible, on peux lui ajouter des fonctionnalité : RubyGems et The Ruby Toolbox.

2.2 Operation Arhitmétique

Opérateur arithmétique : **(exposant), % (modulo) , (+,-,/,*) .

On peux utiliser ruby comme une calculatrice mais ils faut faire attention quand on fait ce genre de chose :

```
$3/2
$=>1  %ruby detecte qu'il travail avec 2 type de class different (entier et d
```

```
$3.0/2.0  %le resultat est celui attendue
$=>1.5
```

Ruby convient très bien lorsqu'il s'agit de travailler avec de très grands et de très petits nombres. Supposez par exemple que vous souhaitez enregistrer quelque part le nombre 192349562563447.

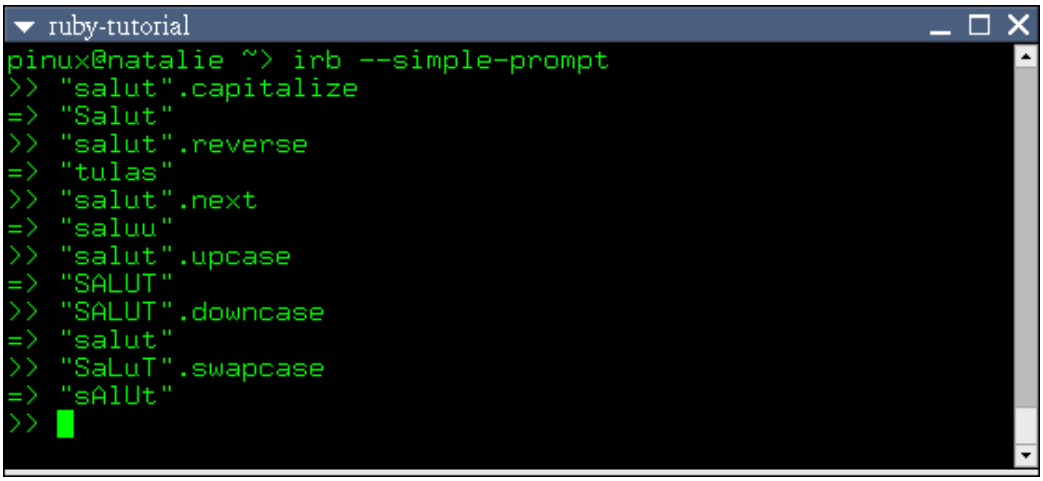
Bon, c'est loin d'être facile à lire. Dans la vie courante, nous sommes habitués à séparer les chiffres par des caractères spéciaux, en fonction du pays d'origine (il s'agit souvent d'une virgule, d'un point ou d'un espace blanc). Par exemple, nos amis anglo-saxons auraient représenté ce fameux nombre de cette façon : 192,349,562,563,447.
Ruby fonctionne exactement de la même façon, en utilisant des "blancs soulignés" (en anglais, "underscores") :

2.3 Chaine de caractère

Les nombres ne sont pas les seuls types de données que vous utiliserez pour programmer avec Ruby. En effet, vous allez probablement devoir manipuler des lettres, des mots ou encore des blocs de texte.

Ruby identifie ce genre de données par le terme chaine de caractères (en anglais, "string"). Voici quelques exemples de chaînes :
-"a"
-"Salut."
-"Longue vie à Ruby!!!"
-"5 est mon nombre préféré. Quel est le votre ?"
-"Snoopy a dit : `#%^?*@!` "

Voici quelque méthode pour manipuler les chaine (il en existe evidemment PLEIN d'autre) :



Voici la liste des classes Ruby de base :

Type de donnée	Classe associée
nombre entier	Integer
nombre décimal	Float
texte	String

Classe	Quelques méthodes
Integer	+, -, /, *, %, **
Float	+, -, /, *, %, **
String	capitalize, reverse, length, upcase, +, *

Voici la liste des quelques methodes effectuant des conversion sur les chaines :

Methode	Description
to_i	string converti en entier
to_i	flottant converti en entier
to_f	string converti en flottant
to_f	entier converti en flottant
to_s	flottant converti en string
to_s	entier converti en string

2.4 Condition if-else-elsif

Peux s'employer avec les operateur de comperaison : **and**,**or**,**not**.

```
age=17
if age < 17
  print 'vous etes mineur'
else if age > 18 and age <= 100
  print 'vous etes majeur'
else
  print 'vous etes un senior'
end
```

Lors de plusieur comparaison dans la condition l'ajout ou non de parenthése peux changer le resultat de la condition, on modifie l'ordre de priorité des opérations. Ex :
(1 == 1 || 2 ==3) 3==4 vaut false
1 == 1 || (2 ==3 3==4) vaux true

La structure **unless**

2.5 Boucle

2.5.1 while

```
compteur = 0

while (compteur <= 4)
  print compteur
  compteur += 1
```

```
end
# => 0 1 2 3 4
```

2.5.2 loop

```
n = 0

loop
 {
  print n
  n += 2
  break if (n > 6)
 }
# => 0 2 4 6
```

2.5.3 until

La boucle until est à la boucle while ce que unless est à if. Cette boucle s'exécute jusqu'à ce qu'une condition soit vraie. Elle est donc totalement équivalente à while !(condition).

```
n = 0

until n>5
  print n
  n += 2
end
# => 0 2 4
```

2.5.4 for

La boucle for exécute des instructions données pour chaque élément d'un ensemble.

```
for n in (0..7)
do (do est facultatif)
  print n," "
end
```

```
# => 0 1 2 3 4 5 6 7
```

cela marche aussi avec de chaine de caractere :

```
for s in "a".."g"
  print n," "
end
# => a b c d e f g
```

2.5.5 Autre maniere de faire des boucles

upto :
Pour la classe Enumerable et String.

```
5.upto(10) {|i| print i, " " }
#=> 5 6 7 8 9 10

#ou

5.upto(10) do |i|
  print i," "
end
#=> 5 6 7 8 9 10
```

Version avec chaine de caratere :

```
"a8".upto("b6") {|s| print s, ' ' }
#=> a8 a9 b0 b1 b2 b3 b4 b5 b6
```

time :
Utilise la classe Time.

```
10.times { puts "hello" }
=>Affichera 10 fois hello
```

each :
Fonctionne avec des collection, classe Array.

```
numbers = [1, 3, 5, 7]
numbers.each #Affichage tableau (marche mais pas recommander)
#ou
numbers.each { |n| puts n } #affichage a chaque ligne
#=> 1 3 5 7
```

2.6 Definition de Variable/Methode/Class

Une variable se declare simplement par son nom, elle ne seras pas typé on peux donc faire :

```
$irb
$var1=7
$=>7
$var2="salut"
```

Les constante se comportent comme des variables sauf qu'a leur déclaration leur nom doit commencer par une majuscule "Constante".
Cela ne permettra pas a la variable de ne pas etre modifie mais indiquera a ruby de nous avertire (avec un message d'erreur) si notre variable est modifié.
Il existe des constante prédéfinie dans ruby tel que : Pi, $\text{Masse}_{e}lectron$, $Distance_{t}erre_{s}oleil$, etc

2.6.1 Méthode

Voici quelque méthode connue prédéfini de Ruby :

-**puts** : Méthode qui affiche une chaine de caractere.
Utilisation : puts(String)
Ex :
nom="Mateo"
puts "Bonjour!!!" + nom + "Le boss"
$=>Bonjour!!! Mateo Le boss

Attention : Si on peux afficher un entier avec puts il faudra le convertir en string avant de pouvoir l'afficher sinon cela provoquera une erreur.Ex :

```
y=30-10
puts "Le resultat de y est " + y          %ERREUR
puts "Le resultat de y est " + y.to_s     %OK
```

2.6.2 Classe

On peux inclure une classe a partir d'un autre fichier rb grace au mot cle "load".
Exemple : load 'nom_fichier' # sans l'extension du fichier .rb

Une classe se definie avec le mot cle "class nom_class" et se fini par "end".

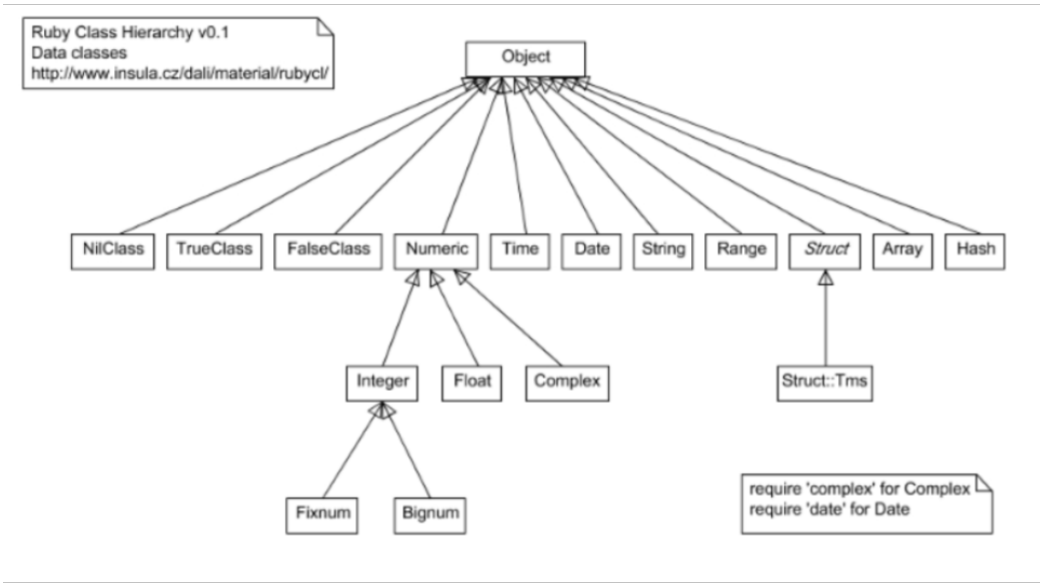
SuperClasse
L'objet de la methode class() method returns the claretourne la classe qui a instancier l'objet.
Toutes les classe en Ruby sont une instance de la Class class.

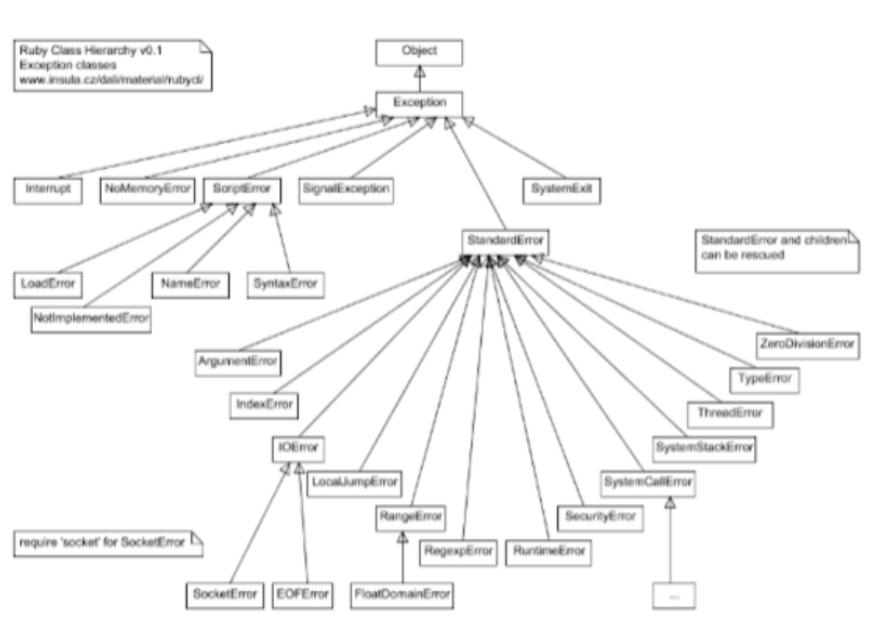
```
Object.class # => Class
class A; end
A.class # => Class
```

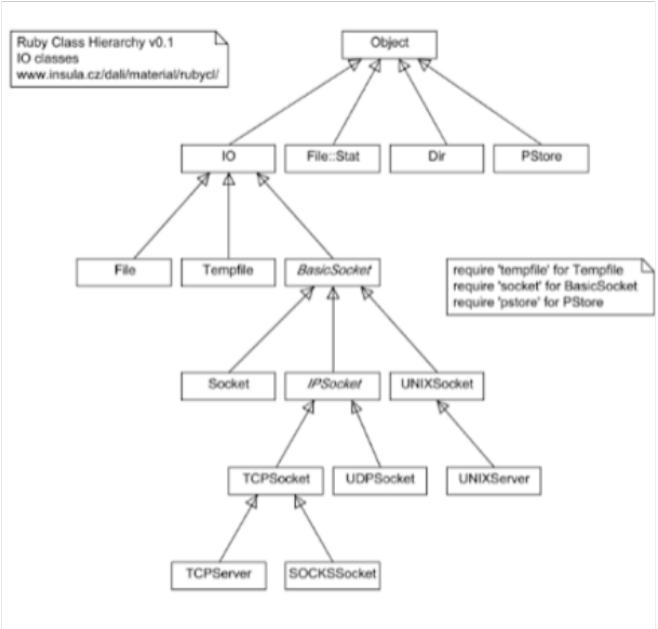
L'instanciation de classe varient pour tout objet en Ruby ce ne sont pas des class objects.

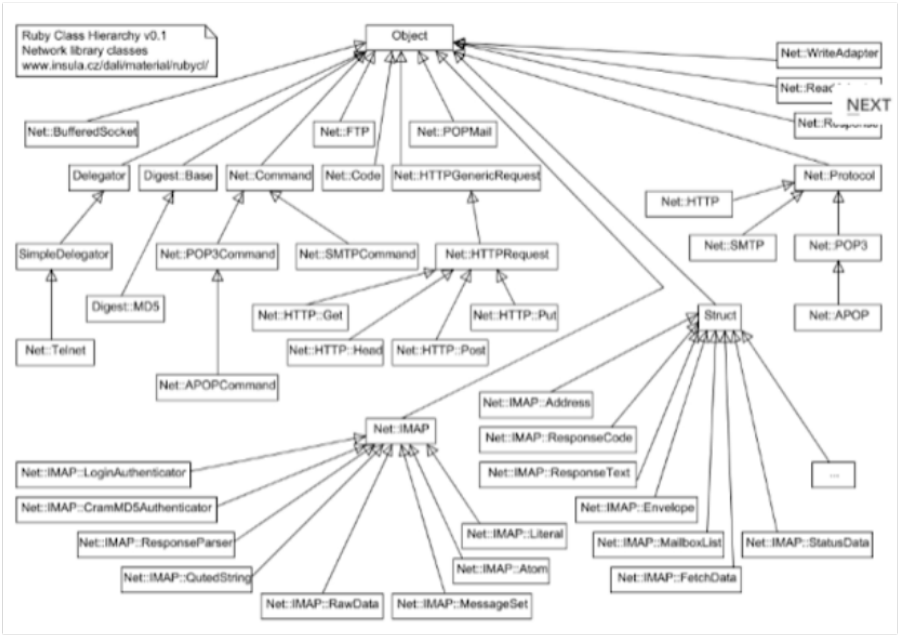
```
class A; end
A.new.class # => A
{}.class # => Hash
4.class # => Fixnum
```

Voici une représentation de la hiearchie des classes en Ruby :


Ruby Class Hierarchy v0.1
Data classes
http://www.insula.cz/dali/material/rubycl/
Object
NilClass
TrueClass
FalseClass
Numeric
Time
Date
String
Range
Struct
Array
Hash
Integer
Float
Complex
Struct::Tms
Fixnum
Bignum
require 'complex' for Complex
require 'date' for Date


Ruby Class Hierarchy v0.1
Exception classes
www.insula.cz/dali/material/rubycl/
Object
Exception
Interrupt
NoMemoryError
ScriptError
SignalException
SystemExit
StandardError
StandardError and children can be rescued
LoadError
NameError
SyntaxError
NotImplementedError
ZeroDivisionError
ArgumentError
TypeError
IndexError
ThreadError
IOError
SystemStackError
LocalJumpError
SystemCallError
RangeError
SecurityError
require 'socket' for SocketError
RegexpError
RuntimeError
SocketError
EOFError
FloatDomainError
...


Ruby Class Hierarchy v0.1
IO classes
www.insula.cz/dali/material/rubycl/
Object
IO
File::Stat
Dir
PStore
File
Tempfile
BasicSocket
require 'tempfile' for Tempfile
require 'socket' for BasicSocket
require 'pstore' for PStore
Socket
IPSocket
UNIXSocket
TCPSocket
UDPSocket
UNIXServer
TCPServer
SOCKSSocket



2.7 Bloc

Utilisation d'accolade ouvrante,fermante bloc. Un bloc permet d'executer une suite d'instruction.
Un bloc bloc peut etre utiliser dans une strucrture de répétition.
Exemple :
0.upto(9) print "salut" => 0.upto(9) do print for "salut". Le bloc agit comme une sorte de parametre a la fonction upto.

Un bloc peut avoir des parametre, déclarer entre "|param|". 0.upto(9) |param| print param.
Le mot clé "yield" permet d'appeler un bloc de code. Exemple :

```
def deux_fois
  yield
  yield
```

```
end

deux_fois {puts "Vive ruby!!"}

Resultat:
Vive ruby!!
Vive ruby!!
```

On peux stocker un bloc dans une variable c'est ce qu'on appele un "objet procédure".

2.8 Collection

Les collection fournissent des services pour manipuler des données qu'elle contient.
La classe Array représente une collection d'items.

2.9 Tableaux

Un tableau peut etre manipuler comme une liste.
Methode sur les tableau : .class, .upcase, .reverse, .sort, .length

Array.new([Taille=0][,remplissage]) Intersection &
Union ||
Concatnation

$Accesaunelement : [1,2,3].at(1) => 2$

$Manipulationd'untableau :$
$- a.firsta.last : Retournelepremier/dernierelement$
$- a.push(val) - > ajoutenfindetableau$
$- a.pop - > retireenfindetableau$
$- a.shift - > retireaudebutdutableau$
$- a.unshift - > ajoutaudebutvaleurdutableau$
$- a.clear : Supprimetoutleselementsdutableau$
$- a.uniq : Supprimelesdoublonsal'affichage$
$-a.uniq! : Supprimelesdoublonsdansletableau(!methodedestructive, adefinirle!faitpartiedu$
$- a.delete(e) : Supprimetoutleselementdutableau$

– $a.delete(e)|e|... : Siaucunelementsupprimrenvoiel'valuationdubloc.$
– $a.delete_at(n), a.delete_if|x|...$