FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

La programmation orientée objet en Java

Licence Professionnelle : Développement Informatique

Pr: Youssef Ouassit

Plan Chapitre 2: Programmations orientée objet (POO)

1. Introduction à POO
2. Concepts fondamentaux de la POO (classe, objet, encapsulation).
3. Déclaration de classes et création d'objets.
4. Constructeurs et mot-clé this.
5. Accès aux membres de la classe (attributs et méthodes).
6. Modificateurs d'accès (public, private, protected).

POO en Java

Premiers concepts orientés objets

Introduction à la POO

Définition:

Un **paradigme de programmation** est un modèle ou une approche permettant de résoudre des problèmes en suivant un ensemble de principes ou de concepts.
Chaque paradigme définit une façon particulière d'organiser et de structurer le code, influençant la manière dont les développeurs pensent et écrivent leurs programmes.

Introduction à la POO

Principaux paradigmes de programmation :

- **Programmation impérative:**

  La programmation impérative est basée sur l'idée de donner des instructions séquentielles à l'ordinateur, comme une série d'ordres. Le code est construit autour de changements d'état à travers des instructions qui modifient directement les valeurs des variables. Ce paradigme inclut l'utilisation de boucles, de conditions, et de fonctions qui décrivent précisément chaque étape.

  Exemples de langages : C, Fortran, Python, Java.

Introduction à la POO

Principaux paradigmes de programmation :

- **Programmation fonctionnelle:**

  La programmation fonctionnelle se concentre sur les **fonctions pures** et l'**absence d'effets de bord**. Elle évite les variables d'état et privilégie l'immuabilité, où les fonctions reçoivent des arguments et renvoient des résultats sans modifier d'autres données.

  Exemples de langages: Haskell, Lisp, Scala, Python, Java(versions récentes)

Introduction à la POO

Principaux paradigmes de programmation :

- **Programmation logique:**

  Basée sur des règles logiques, ce paradigme permet de définir un ensemble de faits et de règles, et le programme déduit des résultats par la logique. Ce type de programmation est couramment utilisé dans les systèmes experts.

  Exemples de langages: Prolog.

Introduction à la POO

Principaux paradigmes de programmation :

- **Programmation déclarative:**

  La programmation déclarative consiste à décrire ce que le programme doit accomplir sans préciser comment. Elle englobe plusieurs paradigmes (comme la programmation fonctionnelle et la programmation logique) et est utilisée dans des contextes comme les bases de données ou les interfaces utilisateur.

  Exemples de langages : SQL, HTML, CSS.

Introduction à la POO

Principaux paradigmes de programmation :

- **Programmation orientée objet (POO)**

  La programmation orientée objet (POO) est un paradigme de programmation qui organise le code autour d'objets, qui sont des instances de classes. Ce paradigme vise à modéliser des concepts du monde réel ou des entités abstraites sous forme d'objets ayant des attributs (données) et des méthodes (comportements).

  Il est centré sur les concepts **d'encapsulation**, **d'héritage**, de **polymorphisme**, et **d'abstraction**.
  Exemples de langages : Java, C++, Python.

Introduction à la POO

Histoire de la POO:

- **Années 1960:** prémices de la POO avec le langage Simula-67
- **Années 1970-1980:** Smalltalk 72, puis Smalltalk 80 marquent le début effectif de la POO ainsi que la pose de ses concepts de base.
- **A partir de 1980:** adoption des langages orientés objet:
  - 1980: Objective C
  - 1983: C++, Common LISP object
  - 1984: Eiffel
  - 1996: java 1
- **Années 1990:** évolutions du concept qui s'étend aux aspects conception, analyse et bases de données.

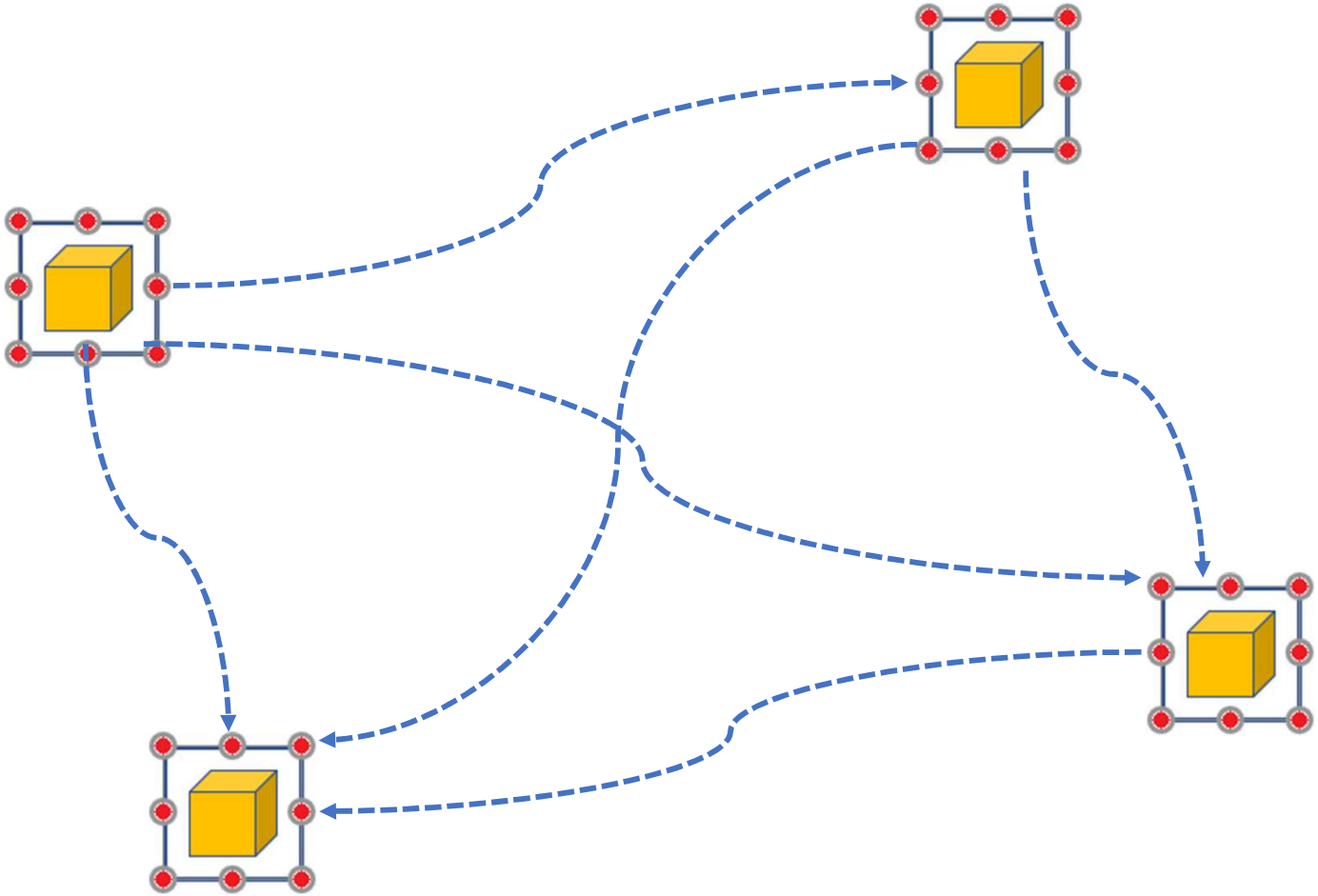
Introduction à la POO

Concepts de la POO:

- Objet
- Class
- Encapsulation
- Héritage
- Polymorphisme
- Abstraction

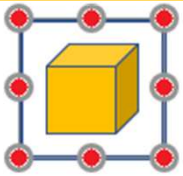
Concepts de la POO

Définition d'un objet



Concepts de la POO

Définition d'un objet



L'objet est une entité qui contient à la fois des données (sous forme **d'attributs**) et des comportements (sous forme de **méthodes**).

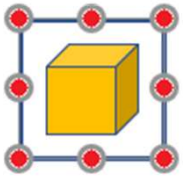
Caractéristiques d'un objet :

1. **Attributs** : Ce sont des variables qui stockent des informations propres à l'objet.
2. **Méthodes** : Ce sont des fonctions associées à un objet qui définissent son comportement.
3. **État** : L'ensemble des valeurs des attributs d'un objet à un moment donné.

Concepts de la POO

Définition d'un objet

Qu'est ce qu'un Objet ?



C1

race = Siamois name = Kati hungry = 2 energy = 9 mood = 7
moew() sleep() play() feed()

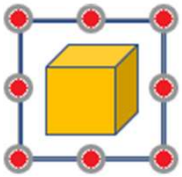
race = Persan name = Mika hungry = 8 energy = 2 mood = 2
moew() sleep() play() feed()

C2

Concepts de la POO

Définition d'un objet

Qu'est ce qu'un Objet ?



C1

race = Siamois
name = Kati
hungry = 2
energy = 9
mood = 7

moew()
sleep()
play()
feed()

hungry ++
energy ++

energy - -
mood ++
meow()

mood ++
energy ++
hungry --

Concepts de la POO

Définition d'un objet

V1

marque = Lucid
annee = 2023
carburant = Electrique
vitesse = 0
niveau_carburant = 53%
Transmission = P

arreter()
demarrer()
accelerer()
ralentir()
charger()

vitesse = 0
transmission = P

vitesse ++
transmission ++

vitesse --
transmission - -

niveau_carburant ++

Concepts de la POO

Définition d'un objet

V1

marque = Lucid
annee = 2023
carburant = Electrique
vitesse = 0
niveau_carburant = 13%
transmission = P

arreter()
demarrer()
accelerer()
ralentir()
charger()

nom = Mohamed
age = 50
ville = Casablanca
voiture = v1

manger()
dormir()
seDeplacer(v1)
changer_voiture()

P1

Concepts de la POO

Définition d'une classe

Une classe est **l'abstraction** d'un ensemble d'objet ayant une structure de **données commune** et disposant des **même mé**thodes.

Une classe est un type permettant de regrouper dans la même structure :

- les informations (**attributs**) relatives à une entité ;
- les procédures et fonctions permettant de les manipuler (**méthodes**).

Concepts de la POO

Définition d'une classe

Classe vs Objet ?

Termes techniques :

- « Classe » est la structure (C'est le type) ;
- « Objet » est une instance de la classe (variable obtenue après instanciation) ;
- « Instanciation » correspond à la création d'un objet

Concepts de la POO

Définition d'une classe

Classe vs Objet ?

race name hungry energy mood
moew() sleep() play() feed()

Classe : Chat

instancier

instancier

C1

race = Siamois name = Kati hungry = 8 energy = 2 mood = 3
moew() sleep() play() feed()

C2

race = Persan name = Mika hungry = 2 energy = 9 mood = 8
moew() sleep() play() feed()

Concepts de la POO

Définition d'une classe

Exercice 1:

Donner la définition d'une classe **Point**, un point est caractérisé par ses coordonnées.

Donner la définition d'une classe **Cercle** qui modélise un cercle. Un cercle est caractérisé par son centre et son rayon.

Ajouter des méthodes utilitaires aux deux classes.

Concepts de la POO

Définition d'une classe

Exercice 1: Solution

Point
- x : Réel - y : Réel
+ deplacer(dx: Réel, dy: Réel)

Cercle
- centre : Point - rayon : double
+ perimetre() + surface() + deplacer(dx: Réel, dy: Réel)

Concepts de la POO

L'encapsulation

Principe :

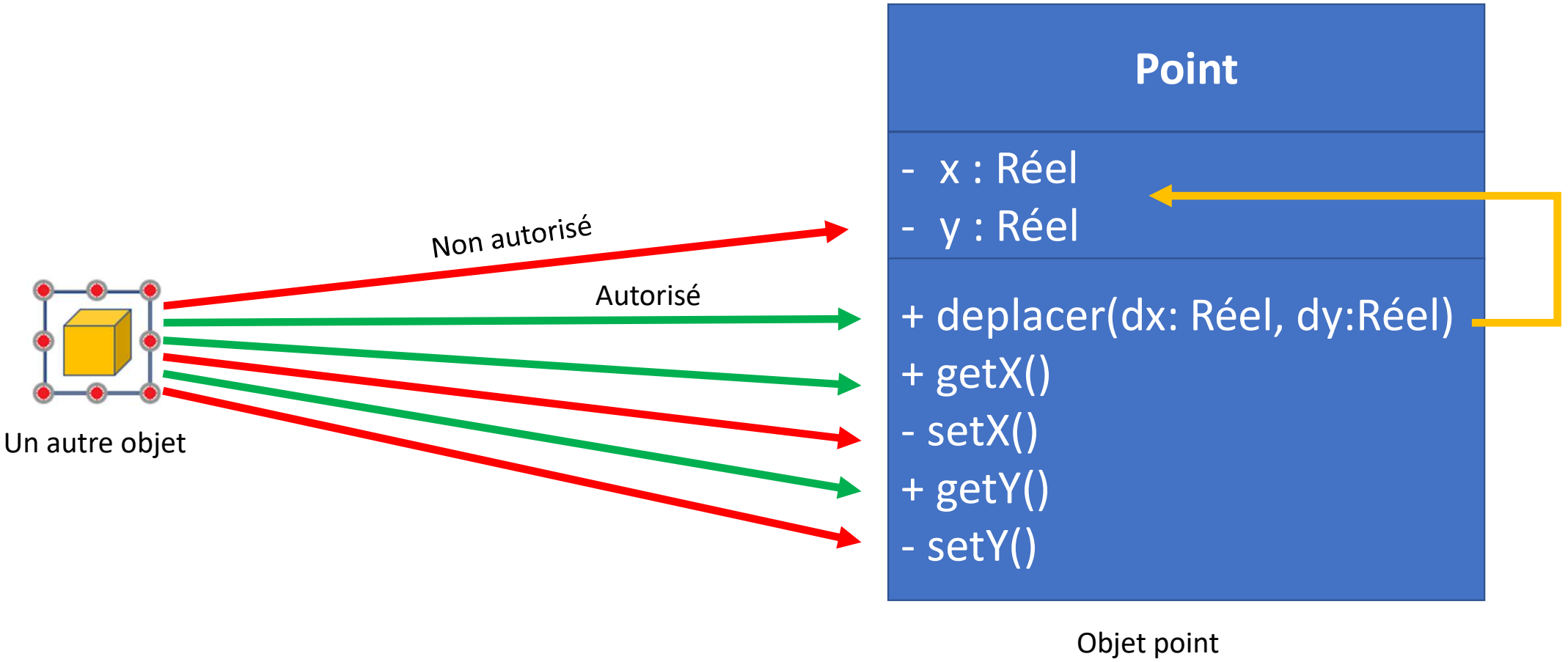
Les méthodes agissent exclusivement sur les données de l'objet.

L'encapsulation est un concept qui concerne les données (attributs) d'un objet.

L'encapsulation (définition): est ce concept de la POO qui interdit l'accès (en lecture/écriture) direct aux données d'un objet, mais qui le permet via des méthodes réservées à cet effet.

Concepts de la POO

L'encapsulation



Objet point

Concepts de la POO

L'encapsulation

Notion de visibilité :

- La notion de visibilité repose sur le principe de l'encapsulation.

- La visibilité sert à garantir la protection des données en :
  - en permettant de masquer les données et certaines méthodes les gérant,
  - et en laissant visibles d'autres méthodes devant servir à la gestion publique de l'objet.

Concepts de la POO

L'encapsulation

Type de visibilités :

- Publique (+)
- Privée (-)
- Protégée (#)
- Visibilité Package (~)

Concepts de la POO

L'encapsulation

Visibilité publique:

- Les attributs et méthodes sont publiques : Cela signifie qu'ils sont accessibles depuis tous les descendants et dans tous les modules.

- Les attributs et méthodes publiques n'ont pas besoin de restriction particulière.

Concepts de la POO

L'encapsulation

Visibilité privée:

- La visibilité privé restreint la porté d'un attribut ou d'une méthode au module où il/elle est déclaré(e).

- Ainsi, si un objet est déclaré dans une unité avec une donnée privée, alors cette donnée ne pourra être accédé qu'à l'intérieur même de l'unité.

L'encapsulation

Visibilité protégée:

- La visibilité 'protégé' correspond à la visibilité 'privé' excepté que tout attribut ou méthode protégé(e) est accessible dans tous les descendants, quel que soit le module où il se situe.

L'encapsulation

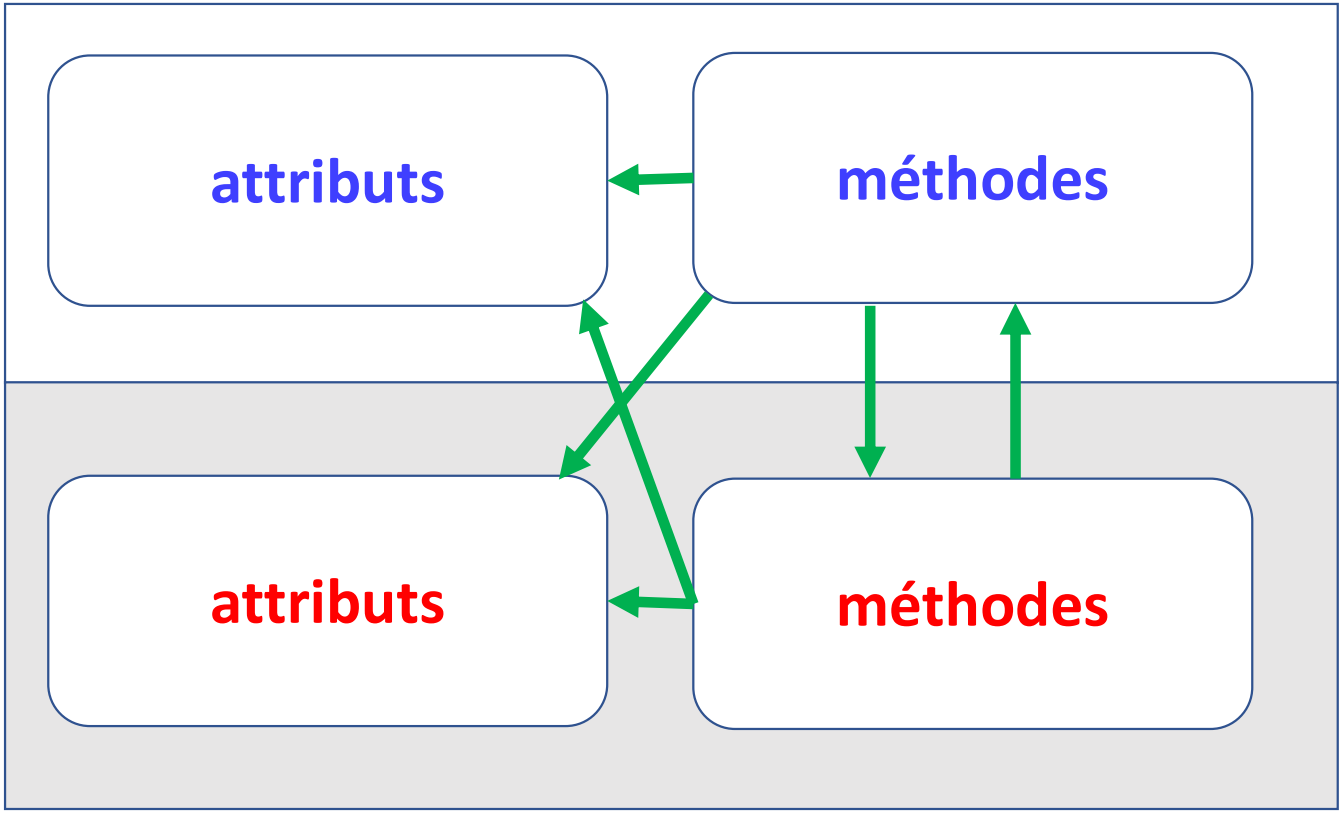
Visibilité package:

- Package (visibilité par défaut) (~) : Accessible uniquement aux classes du même package.

Concepts de la POO

L'encapsulation

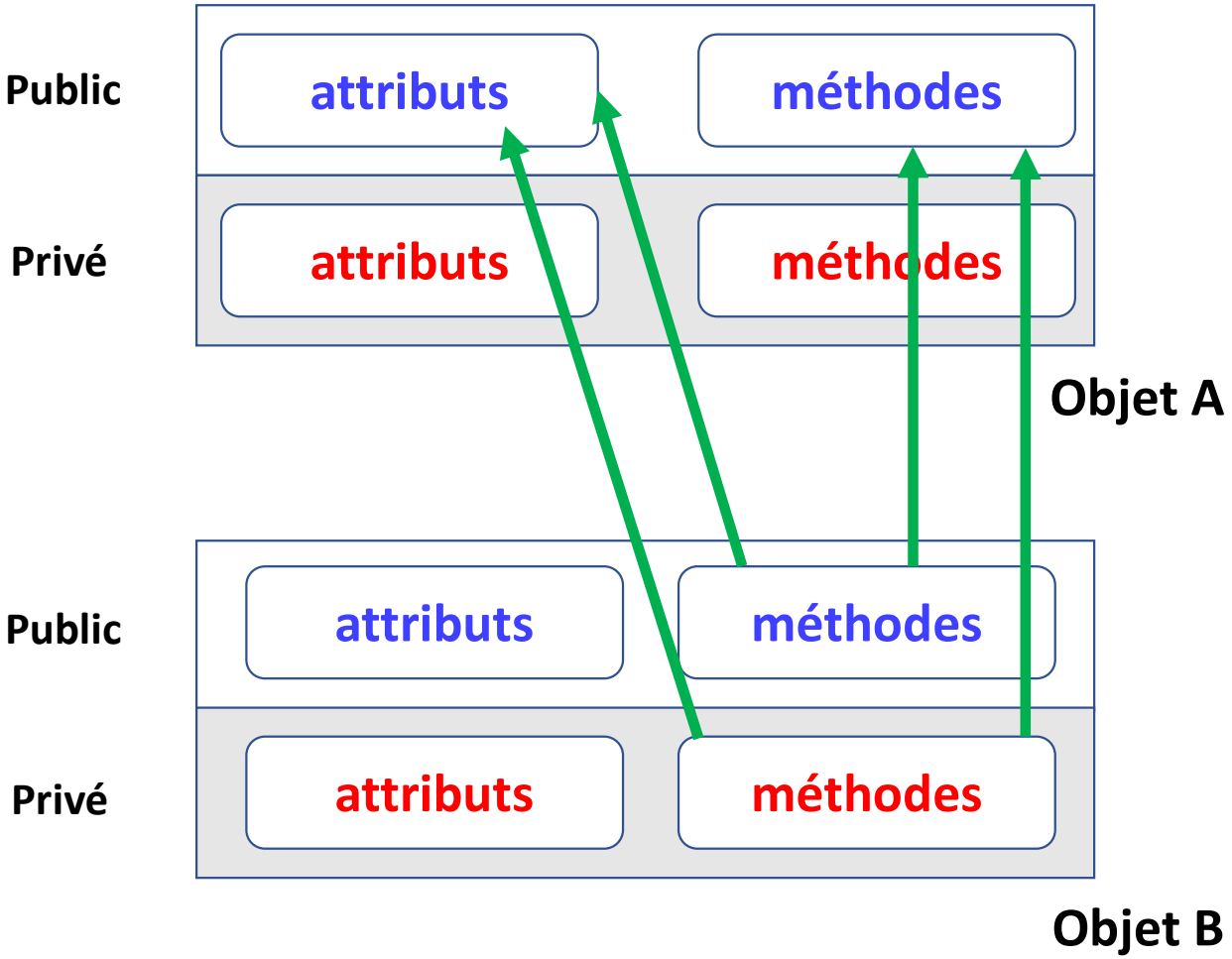
Au sein du même objet :



Concepts de la POO

L'encapsulation

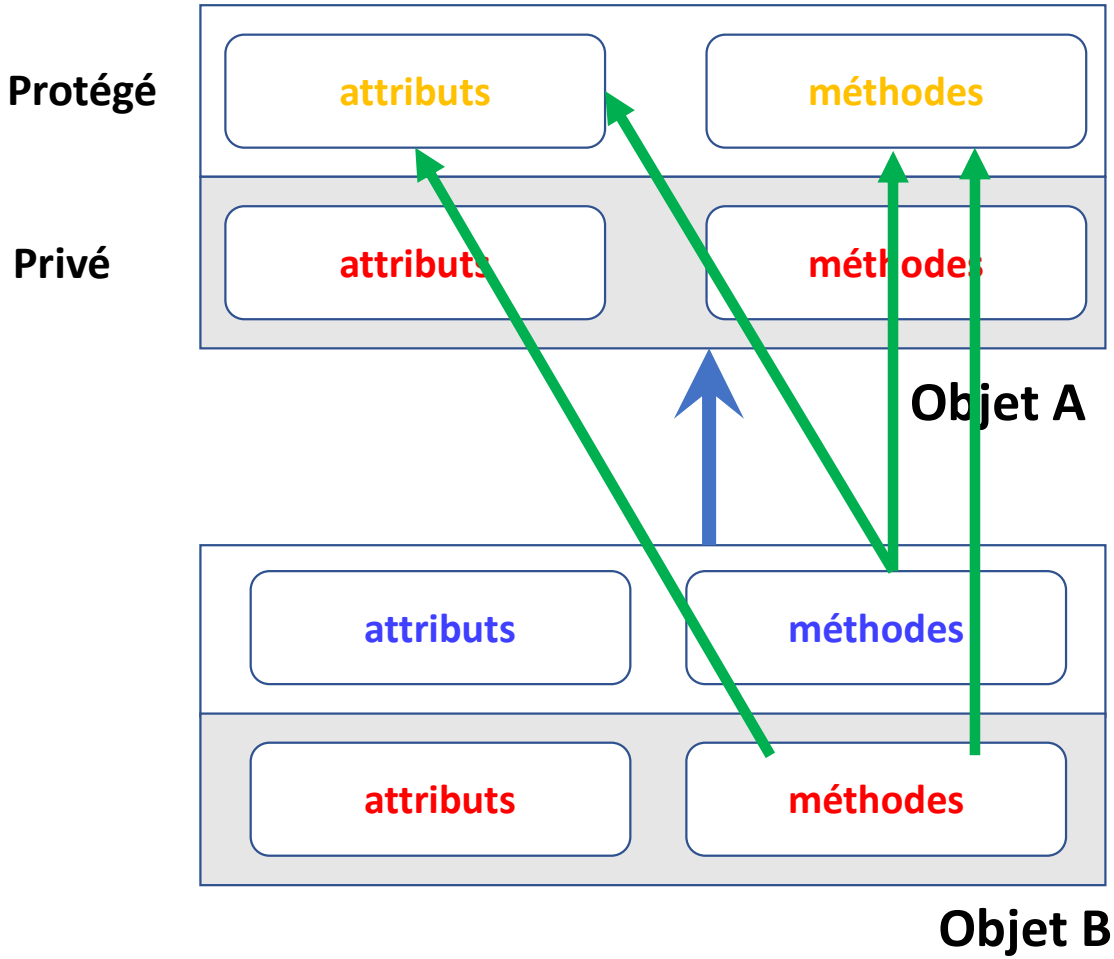
Entre deux objets différents :



Concepts de la POO

L'encapsulation

L'objet A hérite de B :



Concepts de la POO

L'encapsulation

Recommandation:

- Il est recommandé d'utiliser des attributs privés et les rendre accessibles en lecture ou modification via des méthodes publics ou protégés.

Exemple : attribut (age) qui doit être positif.

Concepts de la POO

L'héritage

Définition:

L'héritage est une manière de définir des abstractions de façon incrémentale:

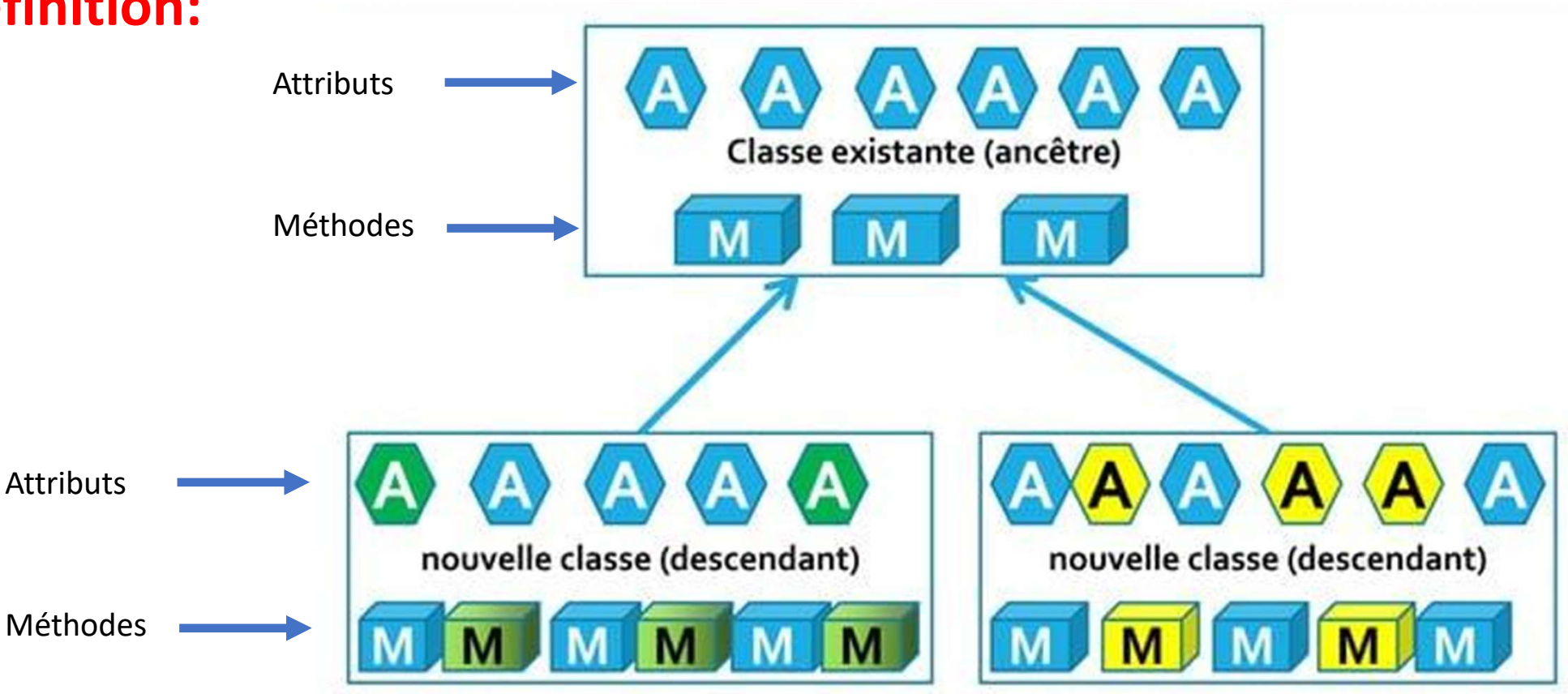
- une définition B peut "hériter" d'une autre définition A.
- la définition B prend A comme base, avec éventuellement des modifications et des extensions.

Héritage (définition): L'héritage en programmation orientée objet (POO) est un mécanisme qui permet à une classe (dite classe fille ou dérivée) d'hériter des propriétés et des méthodes d'une autre classe (dite classe mère ou base). Cela favorise la réutilisation du code et la création d'une hiérarchie entre les classes.

Concepts de la POO

L'héritage

Définition:



Concepts de la POO

L'héritage

Exemples :

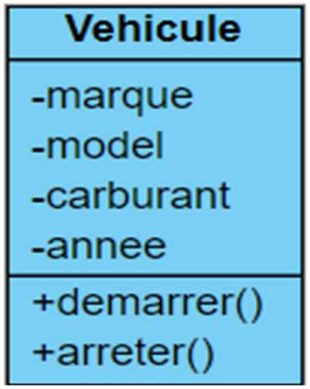
Pour chacun des abstractions suivantes trouver des exemples pour illustrer le concept d'héritage:

- Vehicule
- Animal

Concepts de la POO

L'héritage

Exemples :



Concepts de la POO

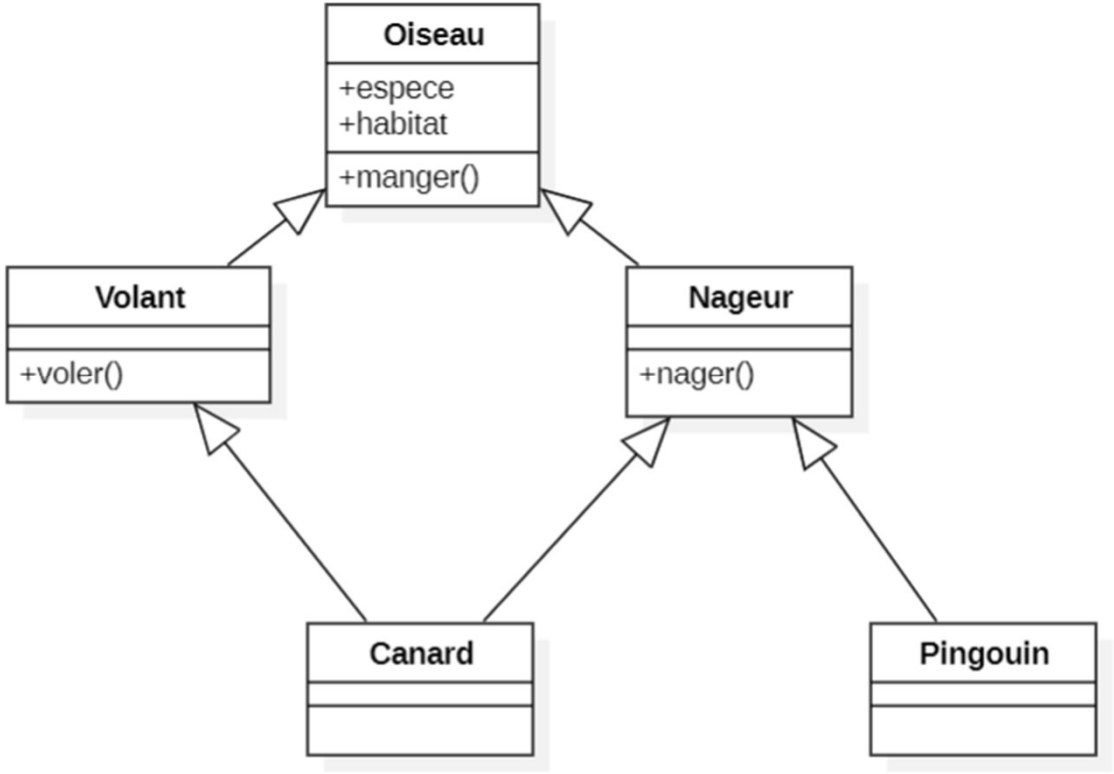
L'héritage

Exemples :

Concepts de la POO

L'héritage

- Quid du multi-héritage?
  - Possible ou non d'un langage à un autre



Concepts de la POO

Polymorphisme

Définition:

Polymorphisme: signifie « plusieurs formes ». En programmation orientée objet, il se manifeste par la capacité d'une seule interface (ou méthode) à s'adapter à différents types d'objets et à produire des comportements différents en fonction de leur type réel.

Plusieurs types :

- Polymorphisme par sous-typage (ou par héritage)
- Polymorphisme paramétrique
- Polymorphisme ad-hoc

Concepts de la POO

Polymorphisme par héritage

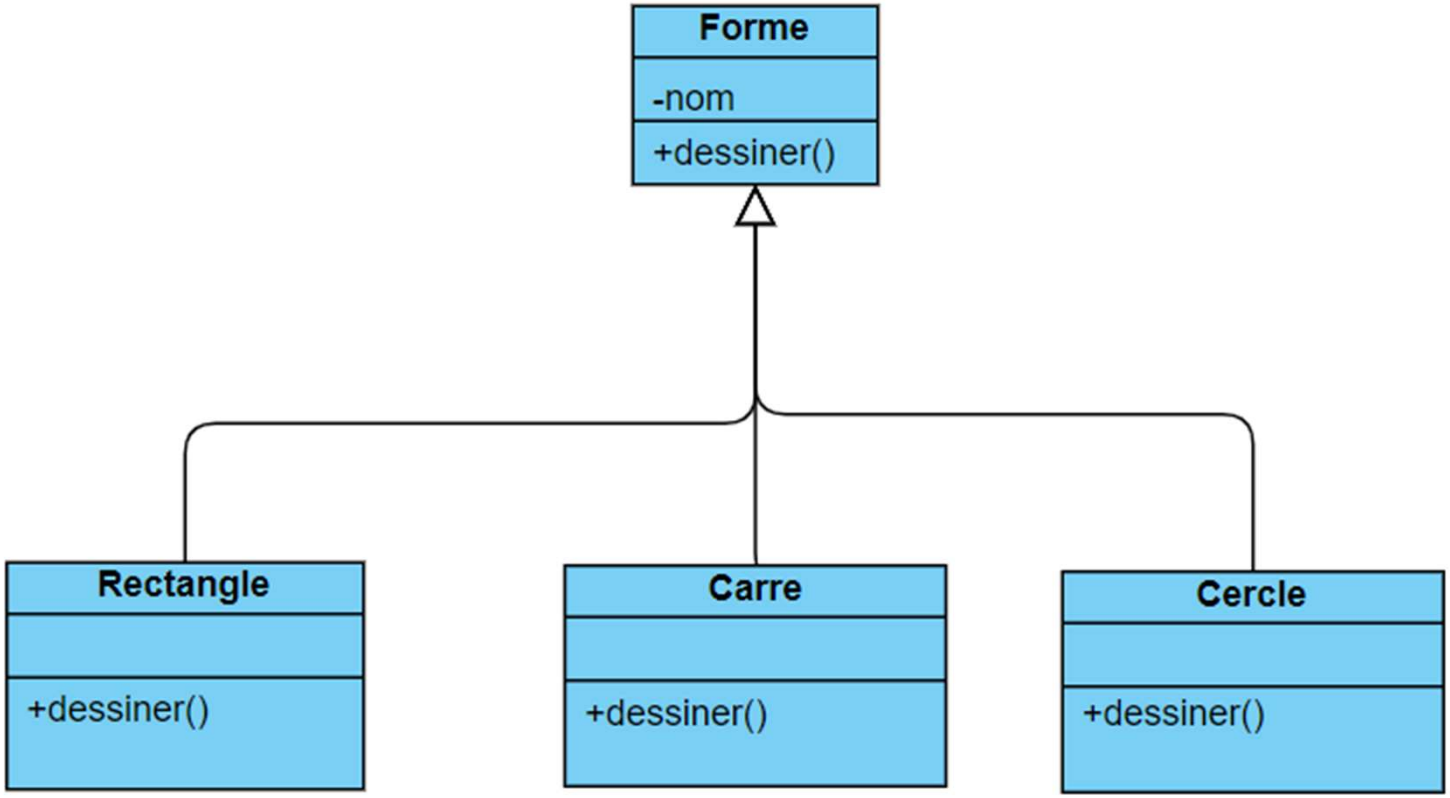
Définition :

Le polymorphisme par héritage est un concept fondamental de la programmation orientée objet (POO) où une classe dérivée (ou sous-classe) peut remplacer ou étendre les comportements définis dans une classe de base (ou super-classe). Ce type de polymorphisme permet d'utiliser des objets de sous-classes à travers une référence de super-classe, offrant une flexibilité et une extensibilité accrues.

Concepts de la POO

Polymorphisme par héritage

Exemple:



Concepts de la POO

Polymorphisme Ad-Hoc

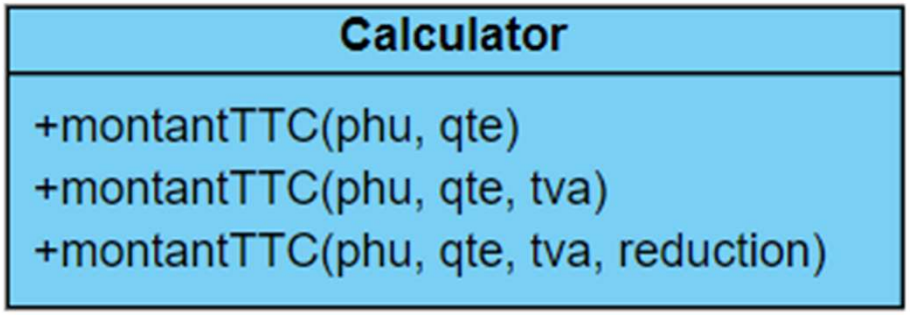
Définition :

Le polymorphisme ad-hoc est un type de polymorphisme qui permet d'utiliser une même fonction ou méthode avec différents types de données, souvent en fonction de la signature (nombre ou type d'arguments) des appels.

Concepts de la POO

Polymorphisme Ad-Hoc

Exemples :



Concepts de la POO

Abstraction

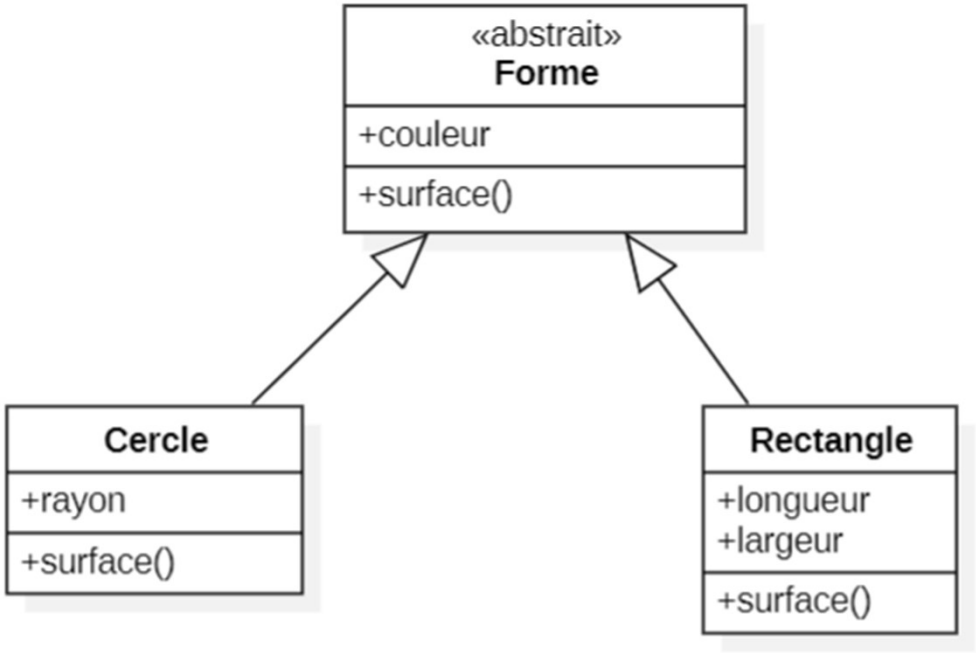
Définition:

L'abstraction est l'un des principes fondamentaux de la programmation orientée objet (POO). Elle consiste à simplifier la complexité d'un système en se concentrant sur les aspects essentiels tout en cachant les détails de l'implémentation. L'objectif est de créer des classes, des objets, et des interfaces qui exposent uniquement les fonctionnalités nécessaires, tout en masquant la logique interne.

Concepts de la POO

Abstraction

Exemple:



Introduction à Java

Programmation orientées objets en Java

Survol du chapitre

- La déclaration des classes
- La création d'objets: Constructeurs et mot-clé « new »
- Les variables: Déclaration et portée
- L'encapsulation: « public », « private » et « protected »
- Les méthodes: Déclaration, interface et surcharge
- Les membres d'instance et de classe: « static »
- Utilisation de l'héritage: « this » et « super »
- Conversion de types
- Polymorphisme
- Classes abstraites
- Interfaces

La déclaration des classes

En Java une classe est déclarée par le mot-clé **class** suivi du nom de la classe.

Il faut aussi spécifier la visibilité de la classe.

```
public class Point
{
      // Déclaration des attributs

      // Déclaration des méthodes
}
```

Les attributs

Déclaration des variables membres (1/6)

- Une variable est un endroit de la mémoire à laquelle on a donné un nom de sorte que l'on puisse y faire facilement référence dans le programme
- Une variable a une valeur, correspondant à un certain type.
- La valeur d'une variable peut changer au cours de l'exécution du programme

Les attributs

Déclaration des variables membres (2/6)

- Rappel: toute variable doit être déclarée et initialisée

- Syntaxe :

[modificateurs] type identificateur [= valeur];

Les attributs

Déclaration des variables membres (3/6)

Les modificateurs d'accès (visibilité) :

1. public : visibilité publique
2. private: visibilité privée
3. protected: visibilité protégée;

Les attributs

Déclaration des variables membres (4/6)

Autres modificateurs :

1. [static]: permet la déclaration d'une variable de classe
2. [final]: empêche la modification de la variable
3. [transient]: on ne tient pas compte de la variable en sérialisant l'objet
4. [volatile]: pour le multithreading

Les attributs

Déclaration des variables membres (5/6)

Exemples :

```
int a;
int a = 0;
public int a;
private int a;
public final float PI = 3.14;
public transient int a = 100;
```

Les attributs

Déclaration des variables membres (6/6)

```
public class Point{

    // les attributs de la classe
    private int x = 0;
    private int y = 0;

    // les méthodes de la classe

}
```

```
public class Point{

    // les attributs de la classe
    private int x = 0, x = 0;

    // les méthodes de la classe

}
```

La création d'objets (1/2)

Le constructeur

- A le même nom que la classe
- Quand un objet est créé, on invoque tout d'abord le constructeur de la classe
- Un constructeur utilise comme arguments des variables initialisant son état interne
- On peut surcharger les constructeurs, i.e définir de multiples constructeurs
- Il existe toujours un constructeur. S'il n'est pas explicitement défini, il sera un constructeur par défaut, sans arguments
- Signature d'un constructeur:
  - Modificateur d'accès (en général public)
  - Pas de type de retour
  - Le même nom que la classe
  - Les arguments sont utilisés pour initialiser les variables de la classe

```
public class Point {

   // les attributs de la classe
   private int x = 0, x = 0;

    // constructeur par défaut
    public Point( ){
        // instructions
    }

    // constructeurs avec deux paramètres
    public Point(int x, int y){
        // instructions
    }
}
```

La création d'objets (2/2)

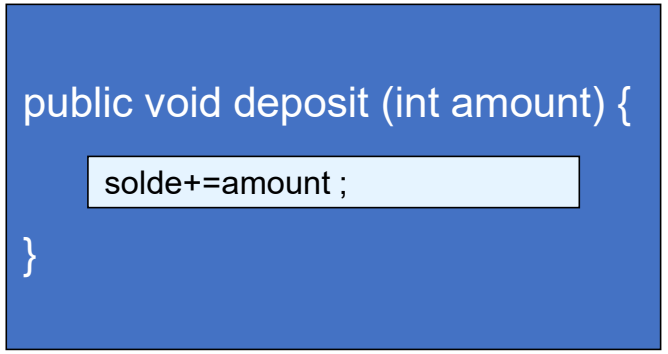
L'appel au constructeur

- Se fait pour initialiser un objet
  - ➔Provoque la création réelle de l'objet en mémoire
  - ➔Par l'initialisation de ses variables internes propres
- Se fait par l'emploi du mot clé « new »

```
Point p1, p2;

p1 = new Point();

p2 = new Point(5, 4);
```

Les méthodes (1/2)

Déclaration d'une méthode



Une méthode est composée de:

- Signature d'une méthode:
  - Modificateurs d'accès : public, protected, private, aucun
  - [modificateurs optionnels] : static, native, synchronized, final, abstract
  - Type de retour : type de la valeur retournée
  - Nom de la méthode (identificateur)
  - Listes de paramètres entre parenthèses (peut être vide mais les parenthèses sont indispensables)
- Au minimum:
  - La méthode possède un identificateur et un type de retour
  - Si la méthode ne renvoie rien ➔ le type de retour est void

Les méthodes (2/2)

La surcharge de méthodes

- La surcharge est un mécanisme qui consiste a dupliquer une méthode en modifiant les arguments de sa signature
- Exemple:

```
Public Account {
        int solde ;
        public void deposit(int amount){
          solde+=amount;
        }
        public void deposit(double amount) {
                solde +=(int) amount;
        }
}
```

L'encapsulation (1/2)

Raisons d'être

Les modificateurs d'accès qui caractérisent l'encapsulation sont justifiées par différents éléments:

- Préservation de la sécurité des données
  - Les données privées sont simplement inaccessibles de l'extérieur
  - Elles ne peuvent donc être lues ou modifiées que par les méthodes d'accès rendues publiques
- Préservation de l'intégrité des données
  - La modification directe de la valeur d'une variable privée étant impossible, seule la modification à travers des méthodes spécifiquement conçues est possible, ce qui permet de mettre en place des mécanismes de vérification et de validation des valeurs de la variable
- Cohérence des systèmes développés en équipes
  - Les développeurs de classes extérieures ne font appel qu'aux méthodes et, pour ce faire, n'ont besoin que de connaître la signature. Leur code est donc indépendant de l'implémentation des méthodes

L'encapsulation (2/2)

Accès aux membres d'une classe

- En java, les modificateurs d'accès sont utilisés pour protéger l'accessibilité des variables et des méthodes.
- Les accès sont contrôlés en respectant le tableau suivant:

Mot-clé	classe	package	sous classe	world
`private`	Y			
`protected`	Y	Y	Y	
`public`	Y	Y	Y	Y
`[aucun]`	Y	Y		

Seul les membres publics sont visibles depuis le monde extérieur.

Une classe a toujours accès à ses membres.

Les classes d'un même package protègent uniquement leurs membres privés (à l'intérieur du package)

Une classe fille (ou dérivée) n'a accès qu'aux membres publics et protected de la classe mère.

Membres d'instance et membres de classe (1/2)

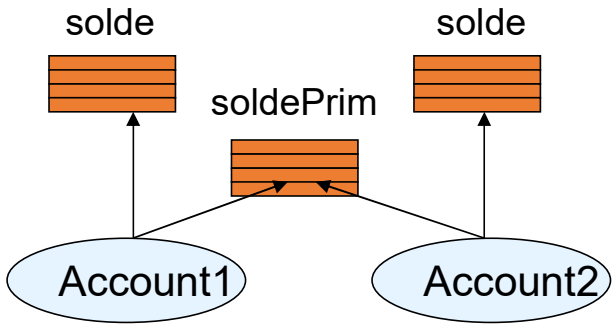
Le mot-clé « static »

- Chaque objet a sa propre "mémoire" de ses variables d'instance
- Le système alloue de la mémoire aux variables de classe dès qu'il rencontre la classe. Chaque instance possède la même valeur d'une variable de classe.

variable d'instance

variable de classe

```
class BankAccount {
        int solde;
        static int soldePrim;
        void deposit(int amount){
                solde+=amount;
                soldePrim+=amount;
        }
}
```

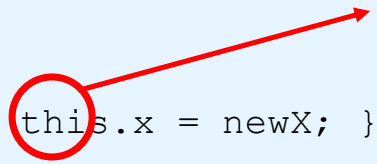


Membres d'instance et membres de classe (2/2)

Le mot-clé « static »

- Variables et méthodes statiques
  - Initialisées dès que la classe est chargée en mémoire
  - Pas besoin de créer un objet (instance de classe)
- Méthodes statiques
  - Fournissent une fonctionnalité à une classe entière
  - Cas des méthodes non destinées à accomplir une action sur un objet individuel de la classe
  - Exemples: `Math.random(), Integer.parseInt(String s), main(String[] args)`
  - Les méthodes statiques ne peuvent pas accéder aux variables d'instances (elles sont "au-dessus" des variables d'instances)

```
class AnIntegerNamedX {
  int x;
  static public int x() { return x; }
  static public void setX(int newX) { this.x = newX; }
}
```

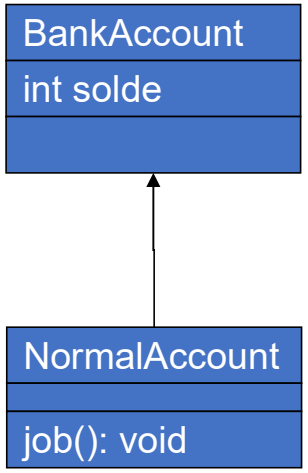


x est une variable d'instance, donc inaccessible pour la méthode static setX

Utilisation de l'héritage (1/5)

- Java n'offre pas la possibilité d'héritage multiple
- La « super super » classe, est la classe *Object* (parente de toute classe)
- Une sous-classe hérite des variables et des méthodes de ses classes parentes
- La clause *extends* apparaît dans la déclaration de la classe

```
class BankAccount {
  protected int solde;
  …
}
class NormalAccount extends BankAccount {
  public void job(){solde+=1000;}
}
```



Utilisation de l'héritage (2/5)

Cacher des données membres

- La variable aNumber du compte normal cache la variable aNumber de la classe générale compte en banque. Mais on peut accéder à la variable aNumber d'un compte en banque à partir d'un compte normal en utilisant le mot-clé super : `super.aNumber`

```
class BankAccount{
        Number aNumber;
}
class NormalAccount extends BankAccount{
        Float aNumber;
}
```

Utilisation de l'héritage (3/5)

Les mots-clé « this » et « super »

- Dans une méthode
  - « this » est une référence sur l'objet en cours lui-même
  - « super » permet d'accéder aux membres de la superclasse (peut être nécessaire en cas de redéfinition, par ex.)
- Dans le constructeur
  - Il existe toujours un constructeur. S'il n'est pas explicitement défini, il sera un constructeur par défaut, sans arguments
  - « this » est toujours une référence sur l'objet en cours (de création) lui-même
  - « super » permet d'appeler le constructeur de la classe parent, ce qui est obligatoire si celui-ci attend des arguments

```
class MyClass{
  int x;
  MyClass(int x){
    this.x=x;// constructeur parent
  }
}
```

```
class Child extends MyClass {
  Child(){
    super(6); // appel du constructeur parent
  }
}
```

Utilisation de l'héritage (4/5)

Les mots-clé « this » et « super »

- En cas de surcharge du constructeur:

```
class Employee {
  String name,firstname;
  Address a;
  int age;
  Employee(String name,String firstname,Address a,int age){
    super();
    this. firstname= firstname;
    this.name=name;
    this.a=a;
    this.age=age;
  }
  Employee(String name,String firstname){
    this(name,firstname,null,-1);
  }
```

Utilisation de l'héritage (5/5)

Redéfinition de méthodes

- La redéfinition n'est pas obligatoire !! Mais elle permet d'adapter un comportement et de le spécifier pour la sous-classe.

```
class BankAccount {
        public void computeInterest(){
                solde+=300;              //annual gift
}}
class NormalAccount extends BankAccount {
        public void computeInterest(){
                super.computeInterest();//call the overriden method
                solde*=1.07;                    //annual increase
}}
```

- Obligation de redéfinir les méthodes déclarées comme abstraites (*abstract*)
- Interdiction de redéfinir les méthode déclarées comme finales (*final*)

Polymorphisme (1/2)

Définition

- Concept basé sur la notion de redéfinition de méthodes
- Consiste à permettre à une classe de s'adresser à une autre en sollicitant un service générique qui s'appliquera différemment au niveau de chaque sous-classe du destinataire du message
- En d'autres termes, permet de changer le comportement d'une classe de base sans la modifier ➔ Deux objets peuvent réagir différemment au même appel de méthode
- Uniquement possible entre classes reliées par un lien d'héritage et suppose un cast « vers le haut » des objets des classes enfants

```
class Bank{
        BankAccount[] theAccounts = new BankAccount[10];
        public static void main(String[] args){
                theAccounts[0] = new NormalAccount("Joe",10000);
                theAccounts[0].computeInterest();
}}
```

Polymorphisme (2/2)

Utilisation du polymorphisme sur des collections hétérogènes

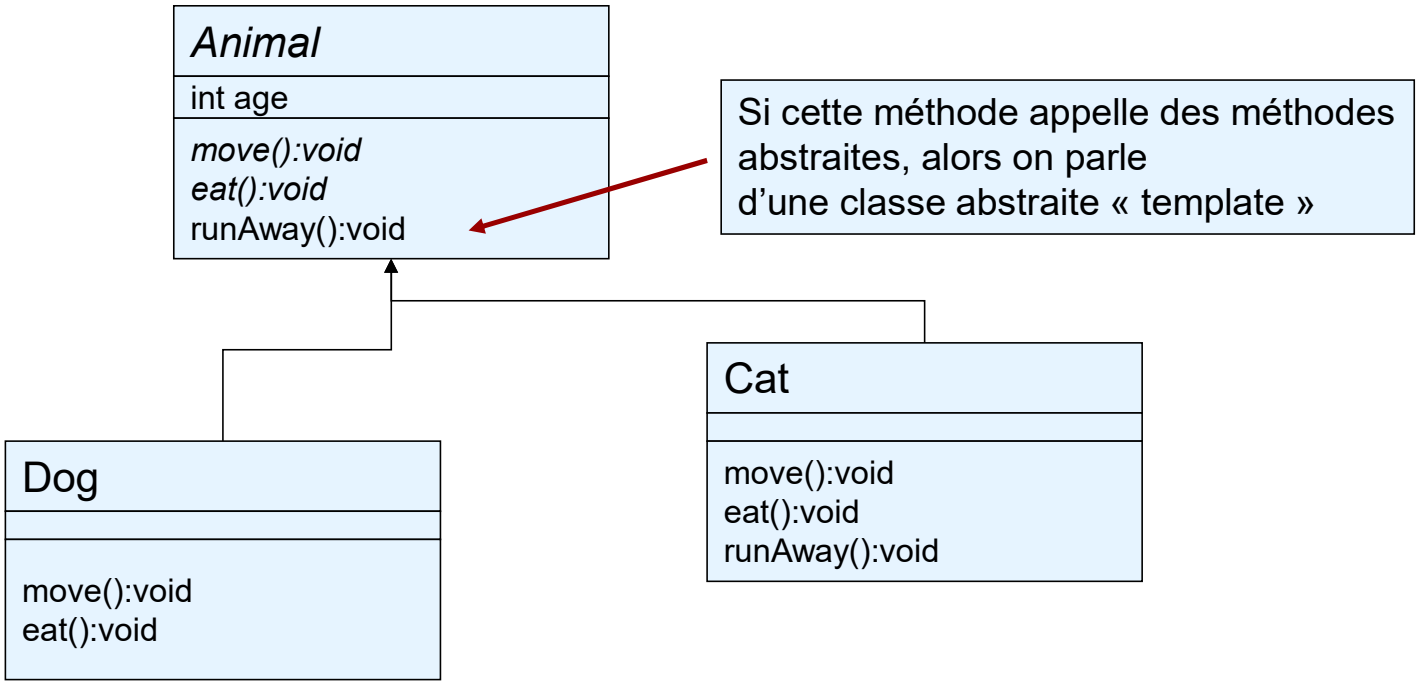
```
BankAccount[] ba=new BankAccount[5];

ba[0] = new NormalAccount("Joe",10000);
ba[1] = new NormalAccount("John",11000);
ba[2] = new SpecialAccount("Jef",12000);
ba[3] = new SpecialAccount("Jack",13000);
ba[4] = new SpecialAccount("Jim",14000);

for(int i=0;i<ba.length();i++)
{
  ba[i].computeInterest();
}
```

Les classes abstraites (1/2)

- Une classe abstraite
  - Peut contenir ou hériter de méthodes abstraites (des méthodes sans corps)
  - Peut contenir des constantes globales
  - Peut avoir des méthodes normales, avec corps
- Une classe abstraite ne peut être instanciée
  - On peut seulement instancier une sous-classe concrète
  - La sous-classe concrète doit donner un corps à toute méthode abstraite
- La déclaration d'une classe abstraite contenant une méthode abstraite ressemble à ceci:

```
abstract class Animal {
   abstract void move();
}
```

Les classes abstraites (2/2)



Les interfaces (1/3)

Définition

- L'interface d'une classe = la liste des messages disponibles = signature des méthodes de la classe
- Certaines classes sont conçues pour ne contenir précisément que la signature de leurs méthodes, sans corps. Ces classes ne contiennent donc que leur interface, c'est pourquoi on les appelle elles-mêmes *interface*
- Ne contient que la déclaration de méthodes, sans définition (corps)
- Permet des constantes globales
- Une classe peut implémenter une interface, ou bien des interfaces multiples

```
public interface Runnable {
  public void run();
}
```

```
public interface GraphicalObject {
  public void draw(Graphics g);
}
```

Les interfaces (2/3)

Raisons d'être

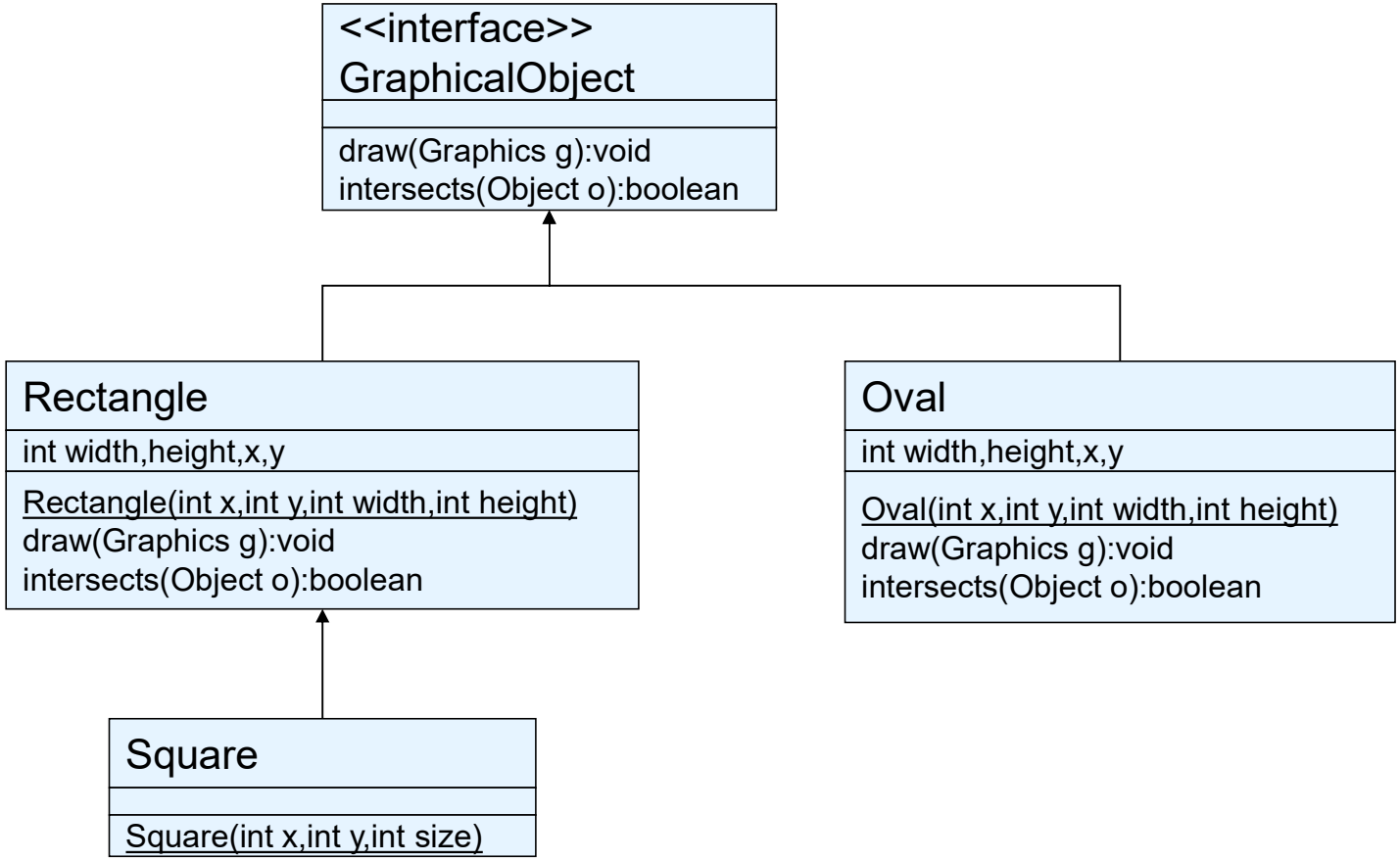
- Forcer la redéfinition / l'implémentation de ses méthodes
- Permettre une certaine forme de multi-héritage
- Faciliter et stabiliser la décomposition de l'application logicielle
- D'une classe qui dérive d'une interface, on dit qu'elle implémente cette interface
- Le mot clé associé est donc logiquement: `implements`

- Exemple:

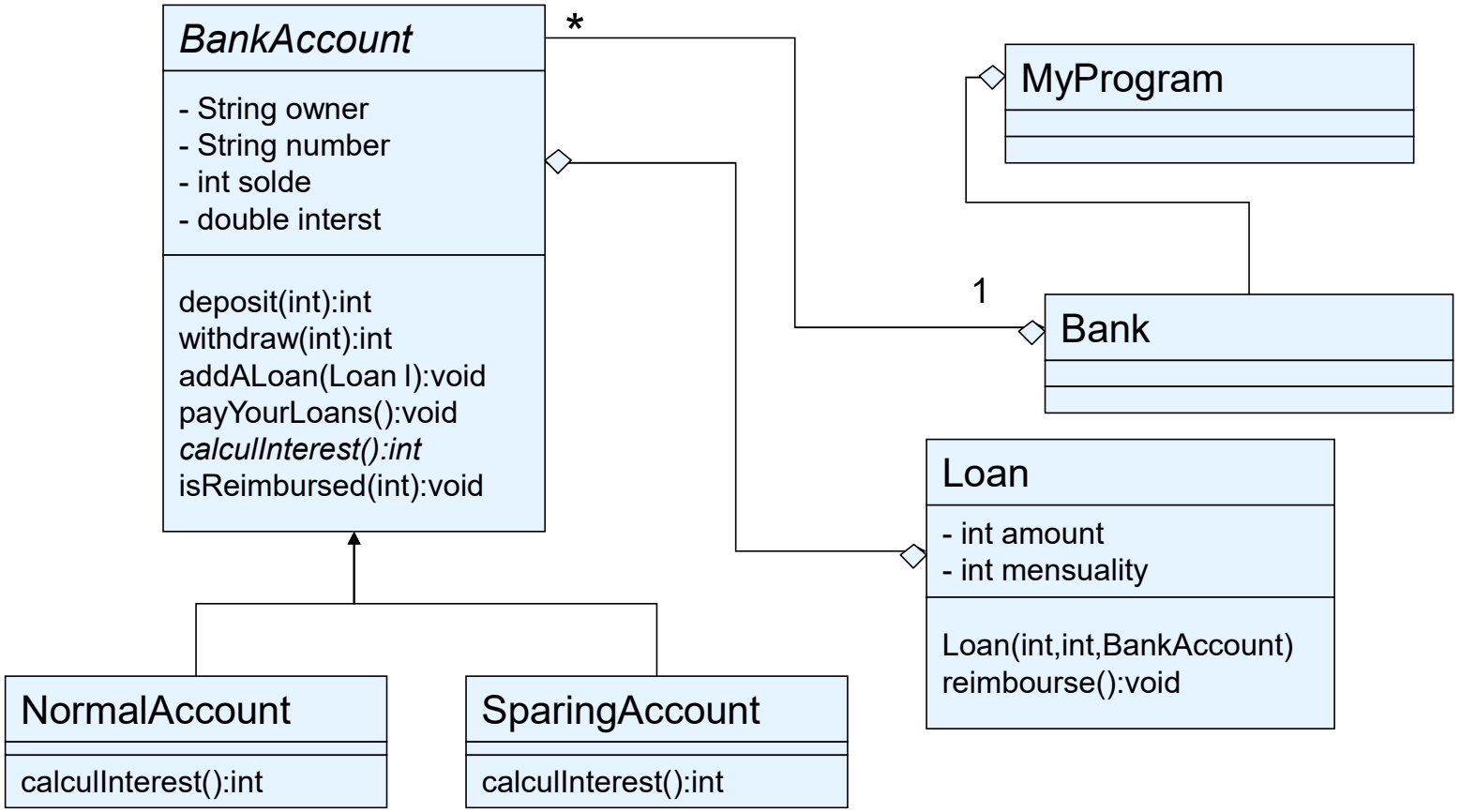
```
public class monApplet extends Applet implements Runnable, KeyListener
```

Les interfaces (3/3)

Exemple



Exercice


BankAccount
- String owner
- String number
- int solde
- double interst
deposit(int):int
withdraw(int):int
addALoan(Loan l):void
payYourLoans():void
calculInterest():int
isReimbursed(int):void
*
1
MyProgram
Bank
Loan
- int amount
- int mensuality
Loan(int,int,BankAccount)
reimbourse():void
NormalAccount
calculInterest():int
SparingAccount
calculInterest():int