Initiation à la programmation orientée objet

UE ModSim

Sébastien Truptil
Ecole des Mines d'Albi-Carmaux

Objectifs du cours

- Présenter les principes, mécanismes et outils de la POO.
- Présenter le lien entre le MOO et le POO.
- Présentation et utilisation des APIs du cours.

Organisation du cours

I. Introduction à la POO

II. Lien entre le POO et le MOO

III. Caractéristiques des POO

IV. Présentation de Java

V. Projet Java

Organisation du cours

I. Introduction à la POO

II. Lien entre le POO et le MOO

III. Caractéristiques des POO

IV. Présentation de Java

V. Projet Java

POO vs Prog. procédurale

- **Programmation procédurale** (exemple : le C)
  - programmes structurés en procédures et fonctions,
  - problèmes en cas de modification de la structures des données,
  - temps d'exécution : très rapide.

- **Programmation Orientée Objet**
  - unité logique : Objet,
  - programmation par «composants»,
  - facilité de l'évolution du code,
  - améliorer la conception et la maintenance des grands systèmes.

différents types de POO

- Il existe deux catégories de langages de POO :
  - les langages à classes,
  - les langages à prototypes.
- chacun pouvant se décomposer sous forme :
  - fonctionnelle (CLOS),
  - impérative (C++, Java, ...),
  - les deux (Python, OCaml…).

Historique de la POO

Historique de la POO

- Les années 60 : les premiers pas de la programmation orientée objet
  - Simula-67, langage de simulation informatique
- Les années 70 : apparition des concepts de base
  - objet, encapsulation, polymorphisme, héritage, ...
- Les années 80 : explosion de l'orientée objet
  - Objective C, C++, Eiffel, ...

Historique de la POO

- Les années 90 : l'âge d'or de la POO
  - Standardisation de C++
  - Apparition du langage Java
- Depuis :
  - stabilisation des langages
  - apparition de nouvelles approches:
    - Analyse par objet
    - Conception orientée objet
    - base de données orientées objets

Organisation du cours

I. Introduction à la POO

II. Lien entre le POO et le MOO

III. Caractéristiques des POO

IV. Présentation de Java

V. Projet Java

Lien entre MOO et POO

«Modéliser c'est construire une représentation d'une entité connue ou inconnue selon des concepts et un vocabulaire communs»

Lien entre MOO et POO

«Modéliser c'est construire une représentation d'une entité connue ou inconnue selon des concepts et un vocabulaire communs»

Mais Surtout, l'étape de modélisation est réalisée indépendamment d'un choix de langage de programmation !!!

Lien entre MOO et POO

«Modéliser c'est construire une représentation d'une entité connue ou inconnue selon des concepts et un vocabulaire communs»

Mais Surtout, l'étape de modélisation est réalisée indépendamment d'un choix de langage de programmation !!!

Par conséquent le lien entre la modélisation orientée objet et la programmation orientée objet se fait par l'objet.

Autrement dit, la MOO et le POO reposent sur le concept d'objet

Lien entre MOO et POO

Lien entre MOO et POO

Définition vue en MOO :

«Un objet est une entité identifiée qui possède un comportement propre dépendant de son état interne et avec lequel il est possible d'interagir»

Lien entre MOO et POO

Définition vue en MOO :

«Un objet est une entité identifiée qui possède un comportement propre dépendant de son état interne et avec lequel il est possible d'interagir»

Un Objet a :

- ✓ Ses propres caractéristiques (Attributs)
- ✓ Ses fonctionnalités (Méthodes)

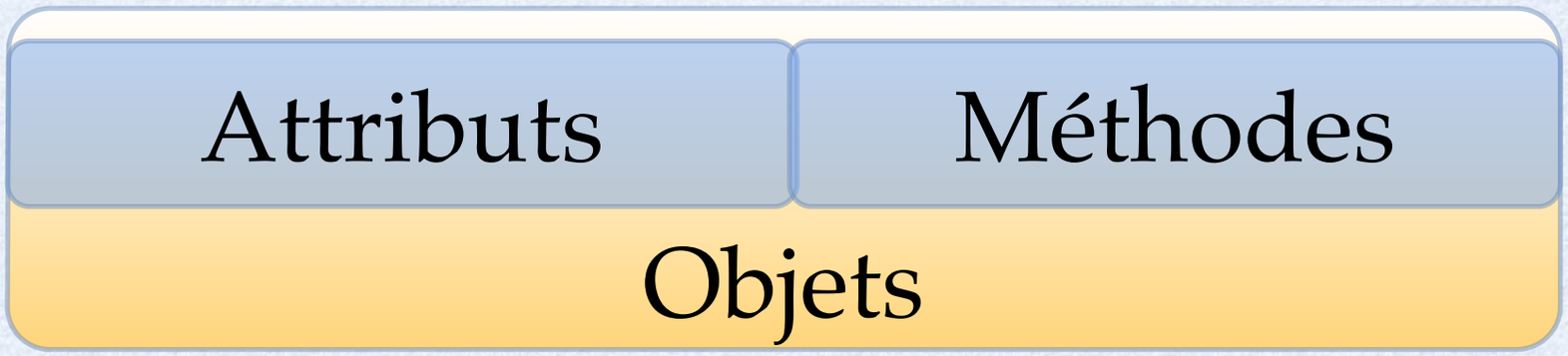
Lien entre MOO et POO

Définition vue en MOO :

«Un objet est une entité identifiée qui possède un comportement propre dépendant de son état interne et avec lequel il est possible d'interagir»

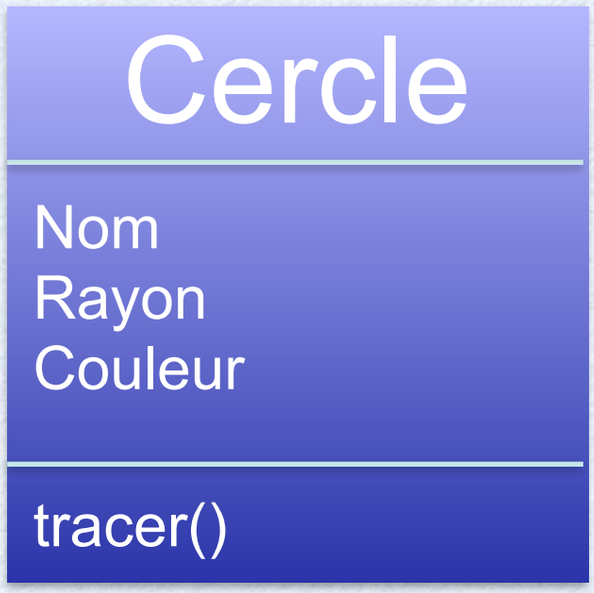
Un Objet a :

- ✓ Ses propres caractéristiques (Attributs)
- ✓ Ses fonctionnalités (Méthodes)

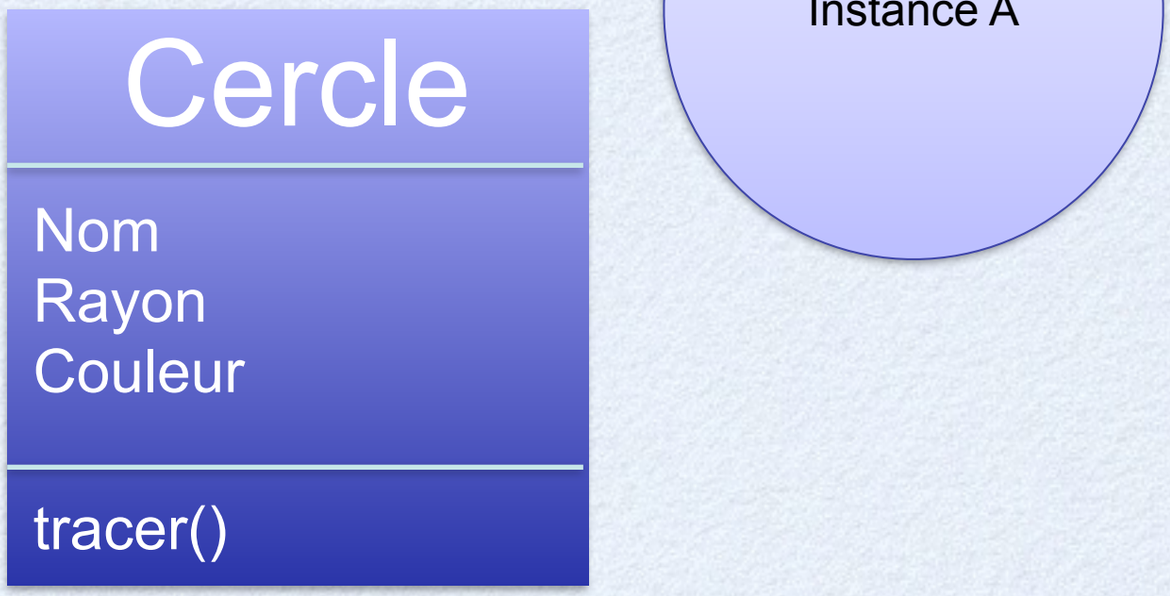


l'Objet

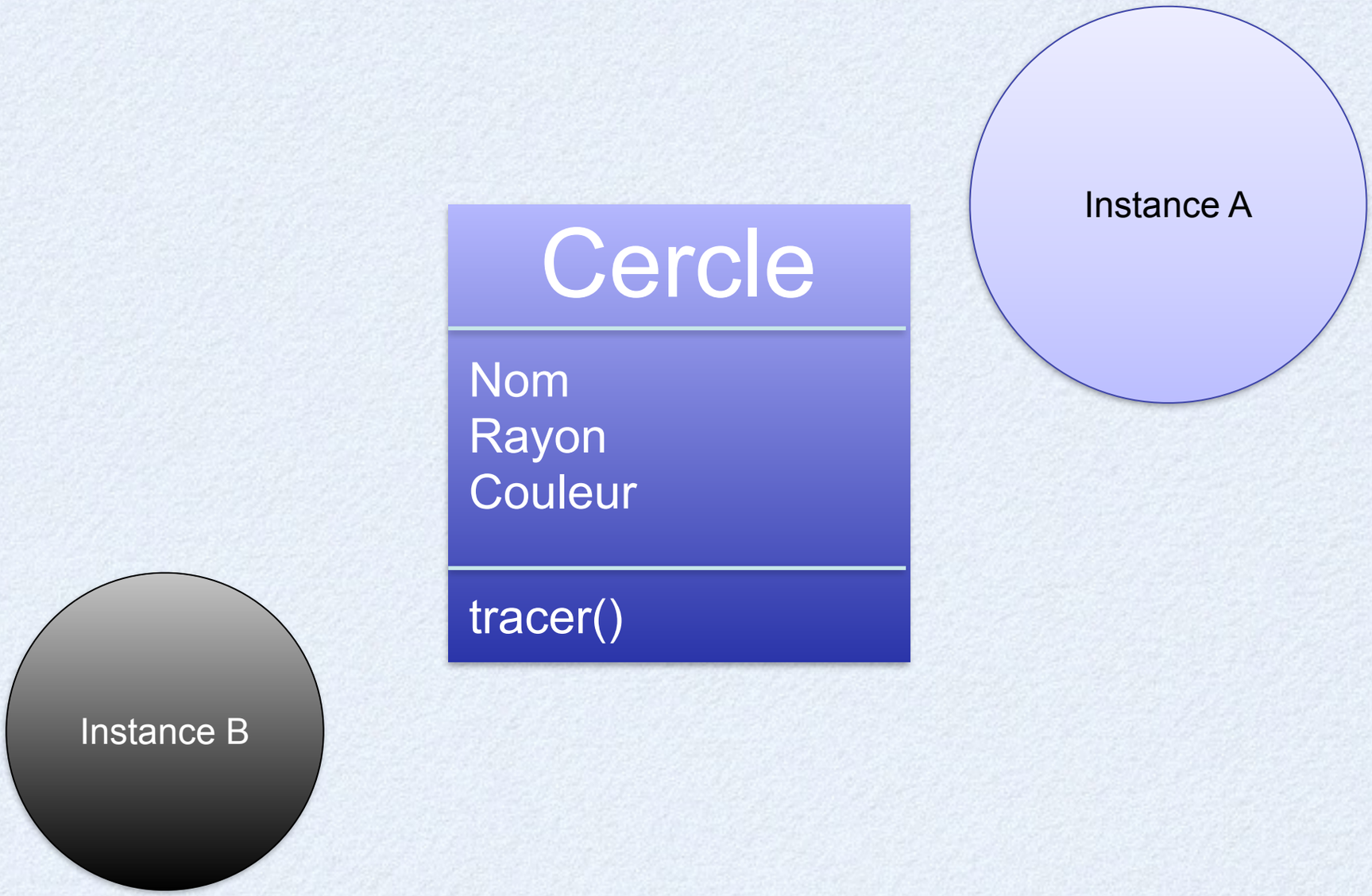
l'Objet



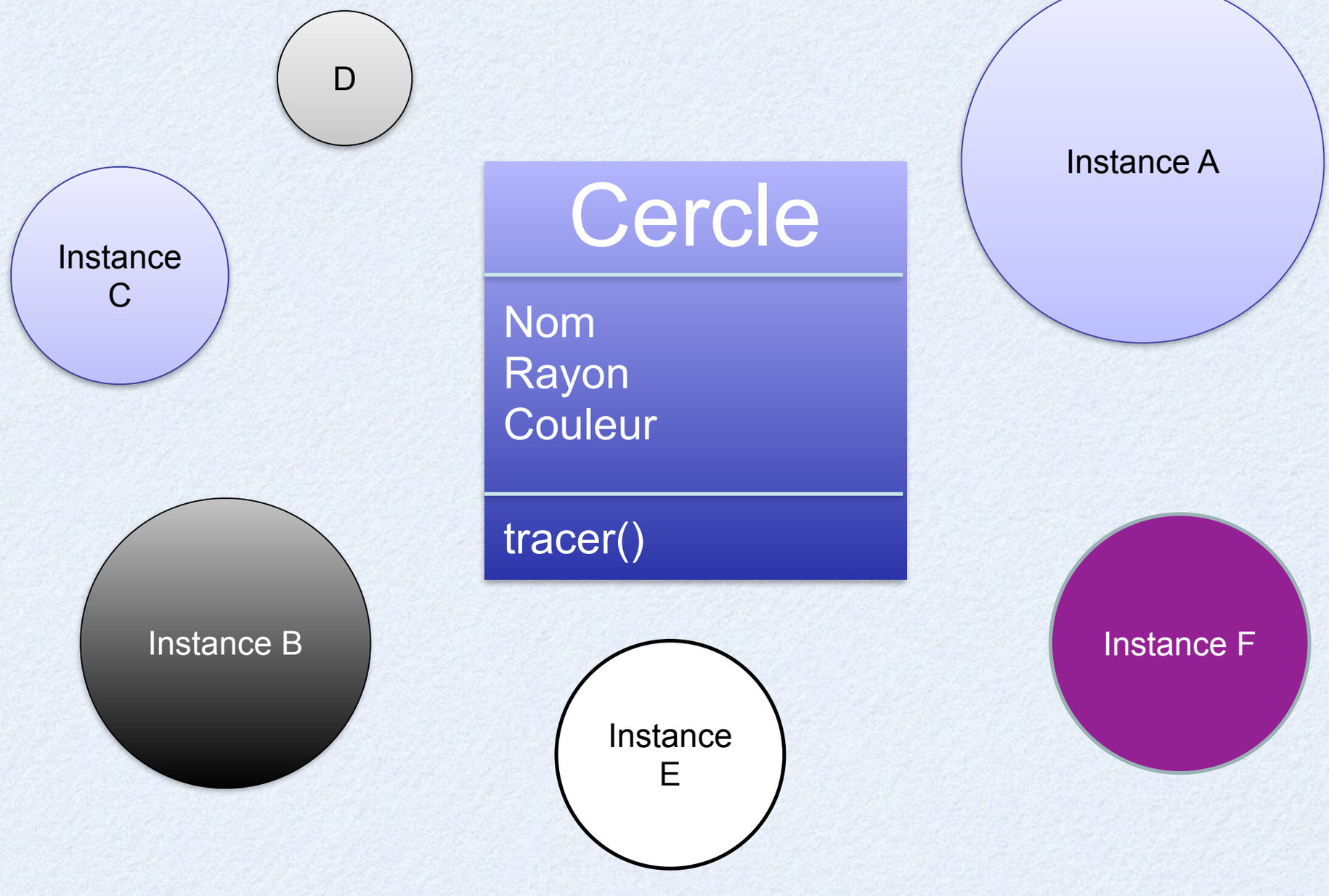
l'Objet



l'Objet



l'Objet


D
Instance A
Instance C
Cercle
Nom
Rayon
Couleur
tracer()
Instance B
Instance E
Instance F

l'Objet

D

Instance A

Instance C

```
public class Circle {

    protected String nom;
    private int rayon;
    public String color;


    public Circle() {
        this.nom = "instance";
        this.rayon = 10;
        this.color = "BLUE";
    }

    public int tracer(){
        return 2*rayon;
}
```

Instance B

Instance F

Instance E

Organisation du cours

I. Introduction à la POO

II. Lien entre le POO et le MOO

III. Caractéristiques des POO

IV. Présentation de Java

V. Projet Java

Caractéristiques des POO : Encapsulation

- Lors de la conception d'un programme orienté-objet, le programmeur doit identifier les objets et les **données** appartenant à chaque objet mais aussi des **droits d'accès** qu'ont les autres objets sur ces données.

Caractéristiques des POO : Encapsulation

- Lors de la conception d'un programme orienté-objet, le programmeur doit identifier les objets et les **données** appartenant à chaque objet mais aussi des **droits d'accès** qu'ont les autres objets sur ces données.

- C'est l'**encapsulation de données**.

Caractéristiques des POO : Encapsulation

- L'**encapsulation de données** dans un objet permet de cacher ou non leur existence aux autres objets du programme. Une donnée peut être déclarée en accès :

Caractéristiques des POO : Encapsulation

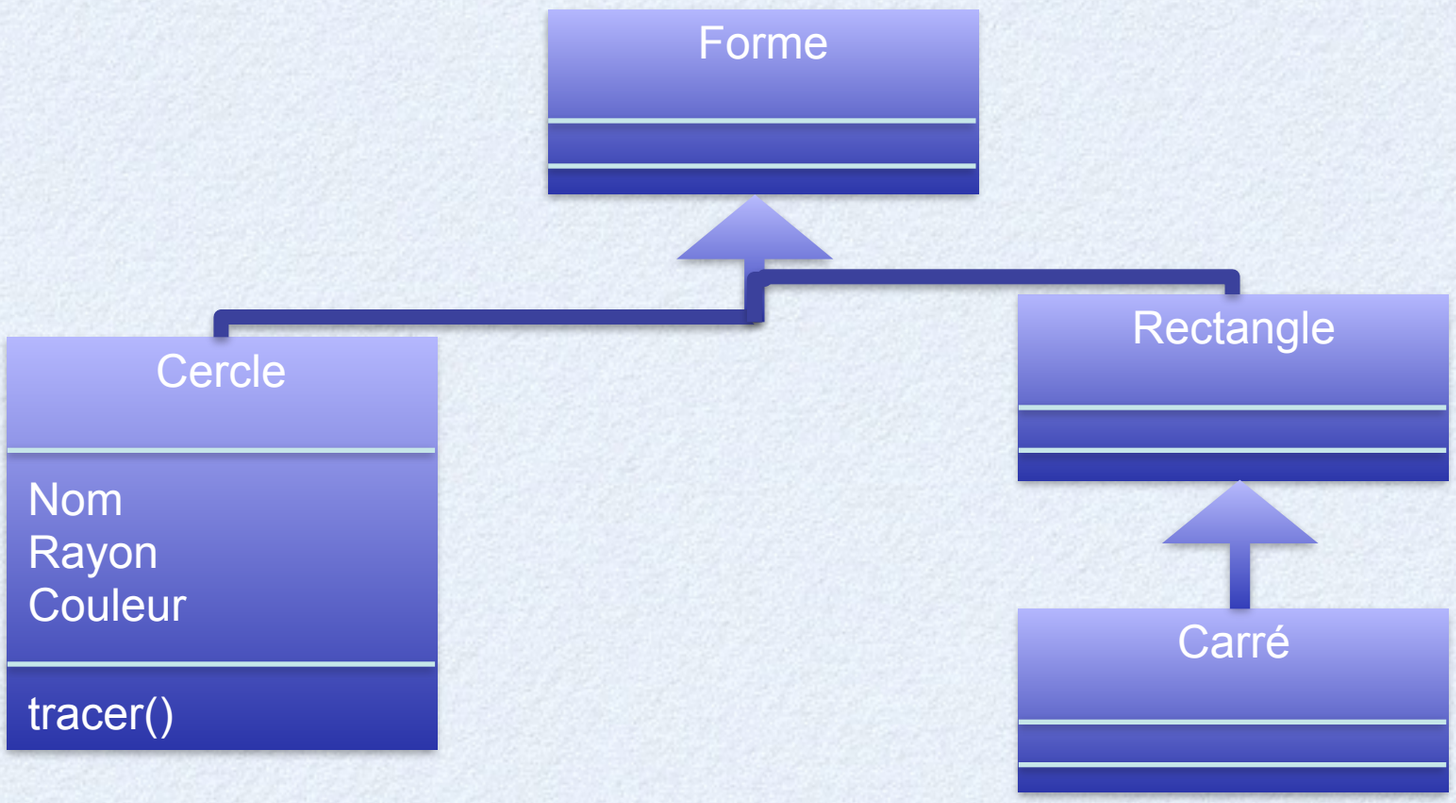
- L'**encapsulation de données** dans un objet permet de cacher ou non leur existence aux autres objets du programme. Une donnée peut être déclarée en accès :

  - **public** : les autres objets peuvent accéder à la valeur de cette donnée ainsi que la modifier ;
  - **privé** : les autres objets n'ont pas le droit d'accéder directement à la valeur de cette donnée (ni de la modifier);
  - **protected** : inaccessible par les objets autres que les classes filles.

Caractéristiques des POO : Héritage

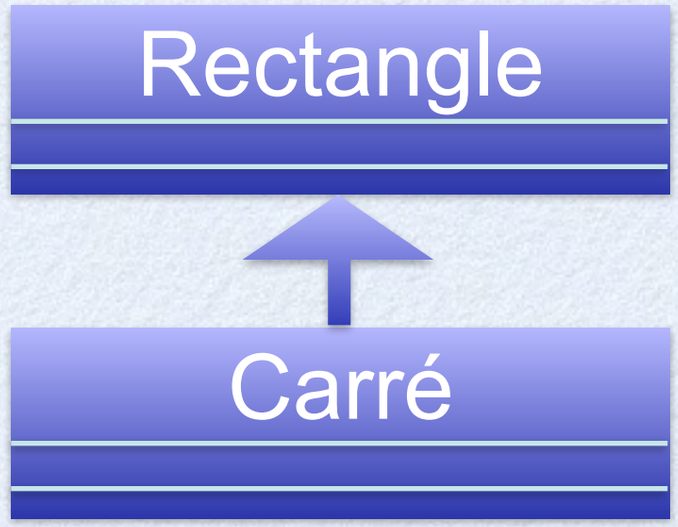
Définition vue en MOO :

L'héritage permet de décrire les notions de spécialisation ou de généralisation.



Caractéristiques des POO : Héritage

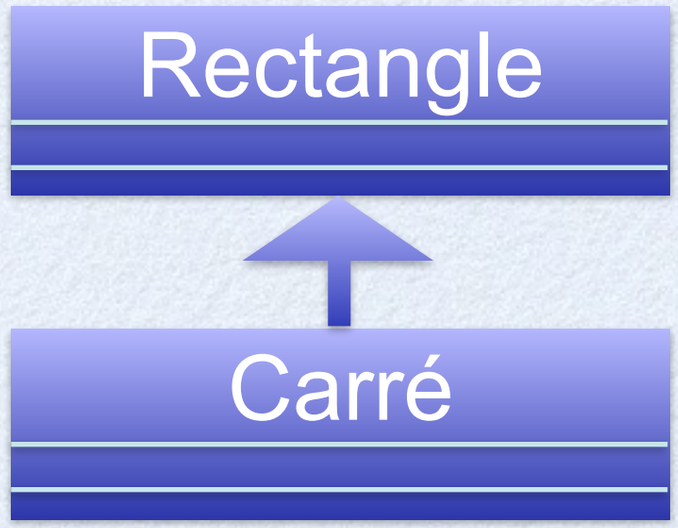
- Deux types d'héritage
  - L'héritage simple :



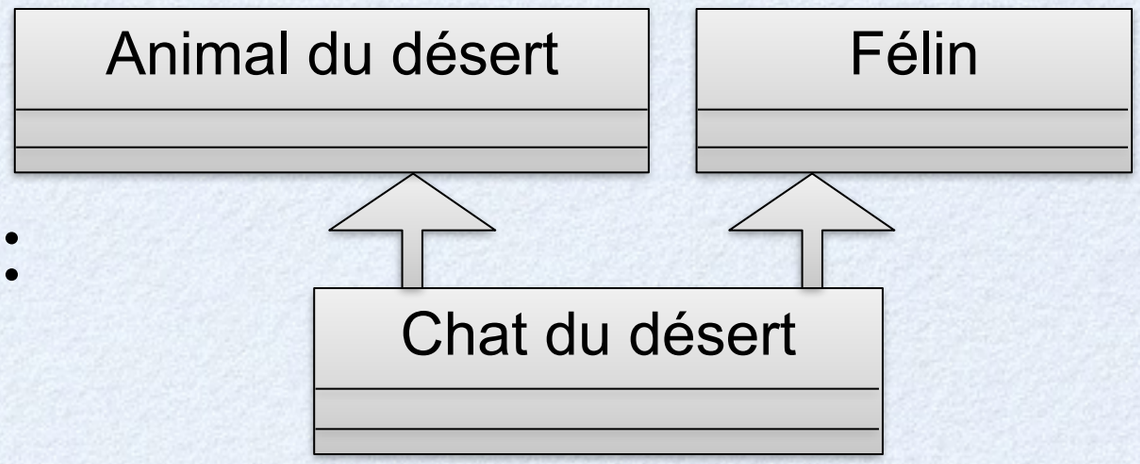
  - L'héritage multiple :

Caractéristiques des POO : Héritage

- Deux types d'héritage
  - L'héritage simple :

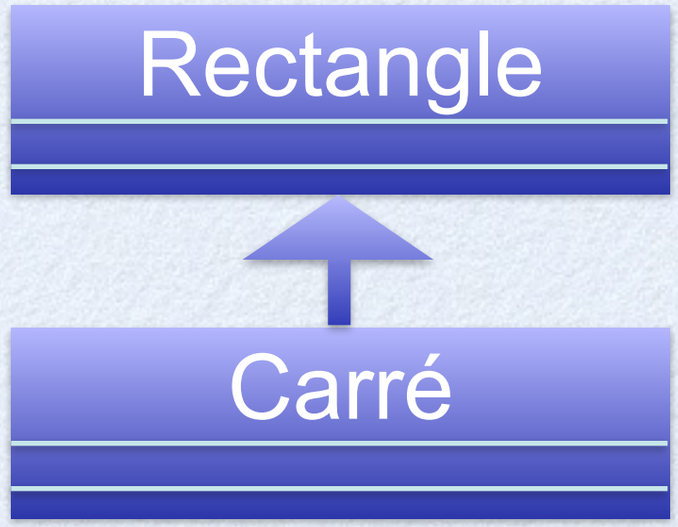


  - L'héritage multiple :



Caractéristiques des POO : Héritage

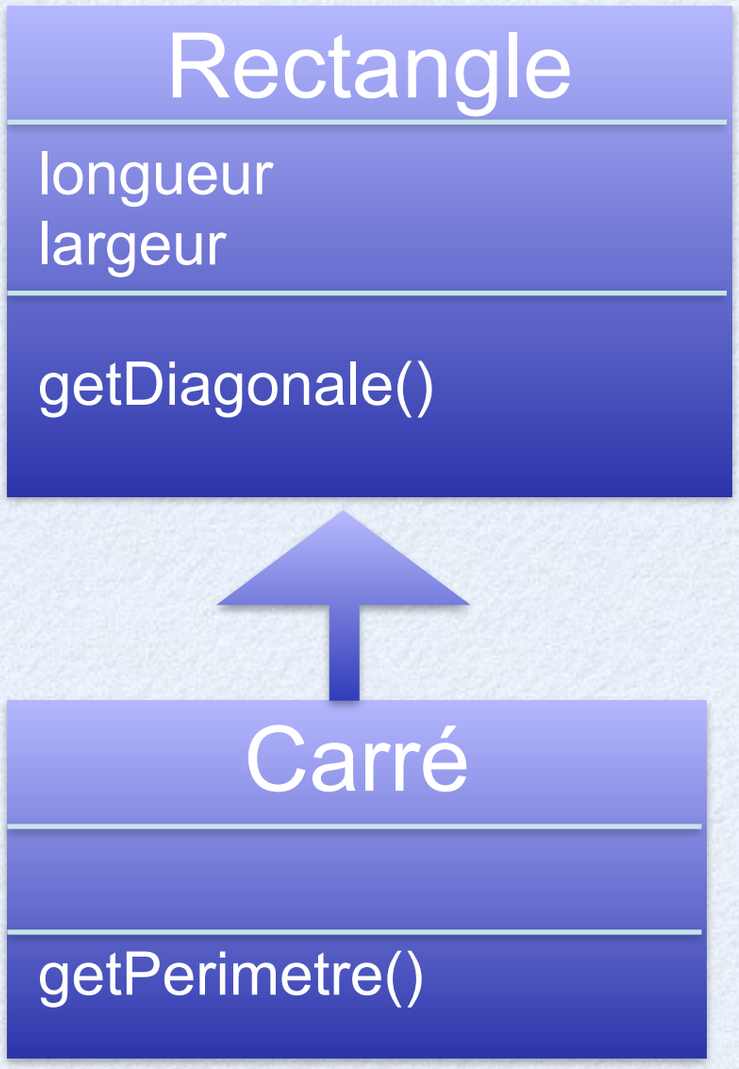
- Deux types d'héritage
  - L'héritage simple :



  - L'héritage multi

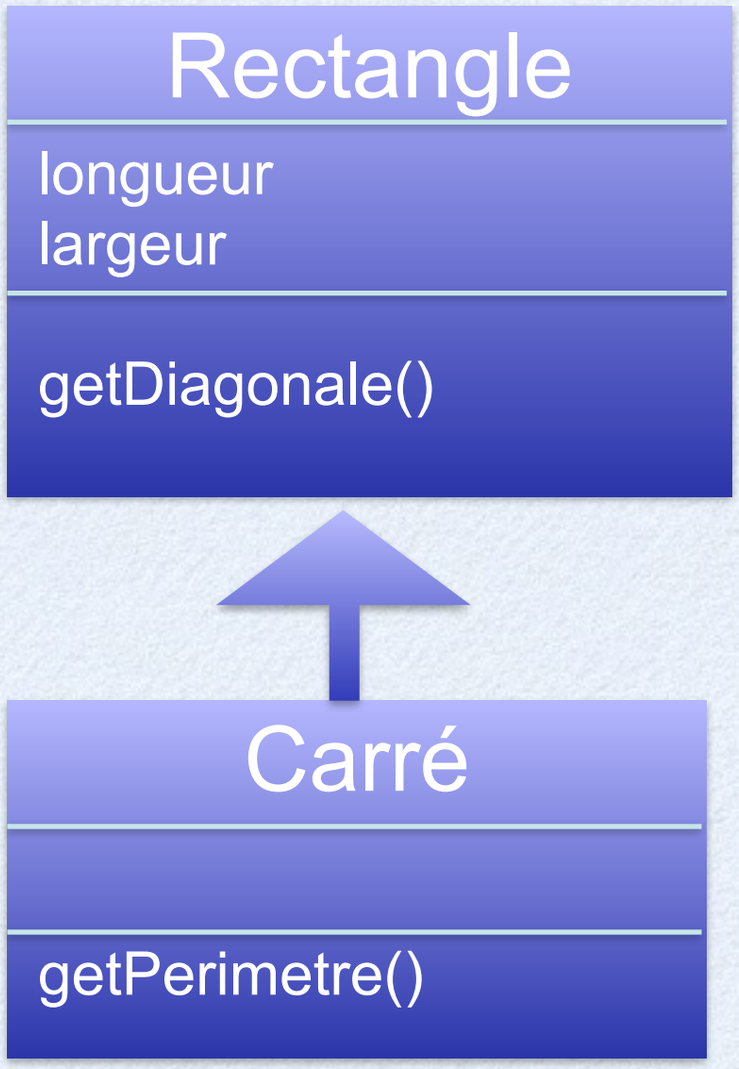
Caractéristiques des POO : Héritage

- L'héritage simple :



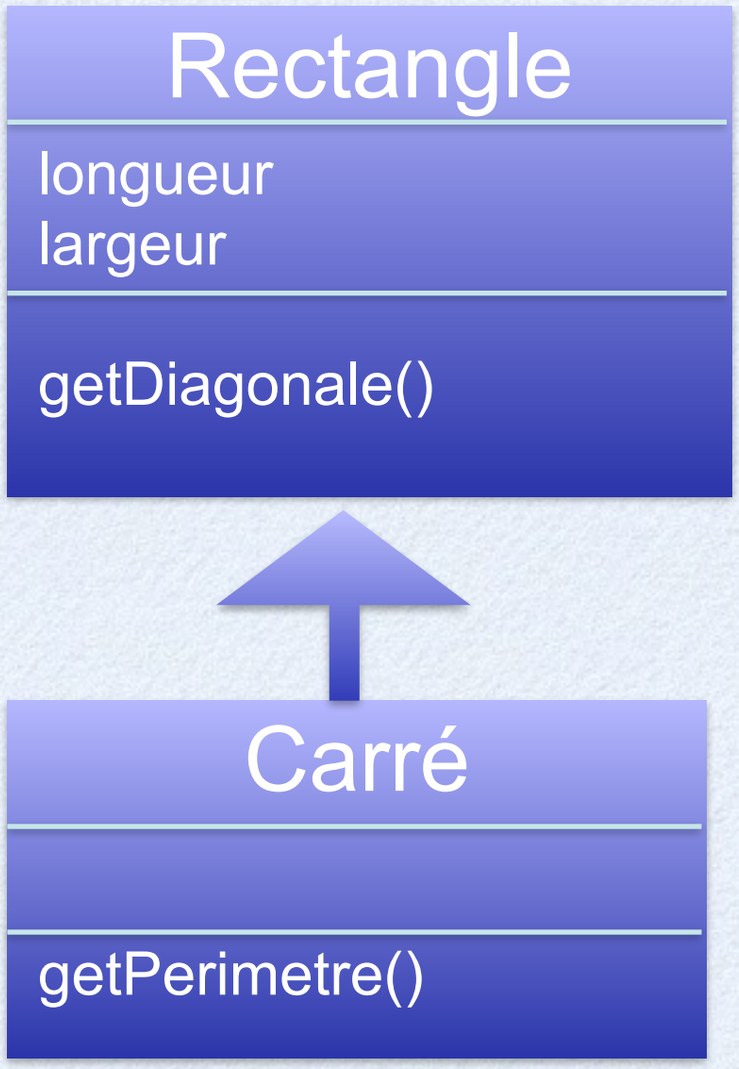
Caractéristiques des POO : Héritage

- L'héritage simple :



Caractéristiques des POO : Héritage

- L'héritage simple :



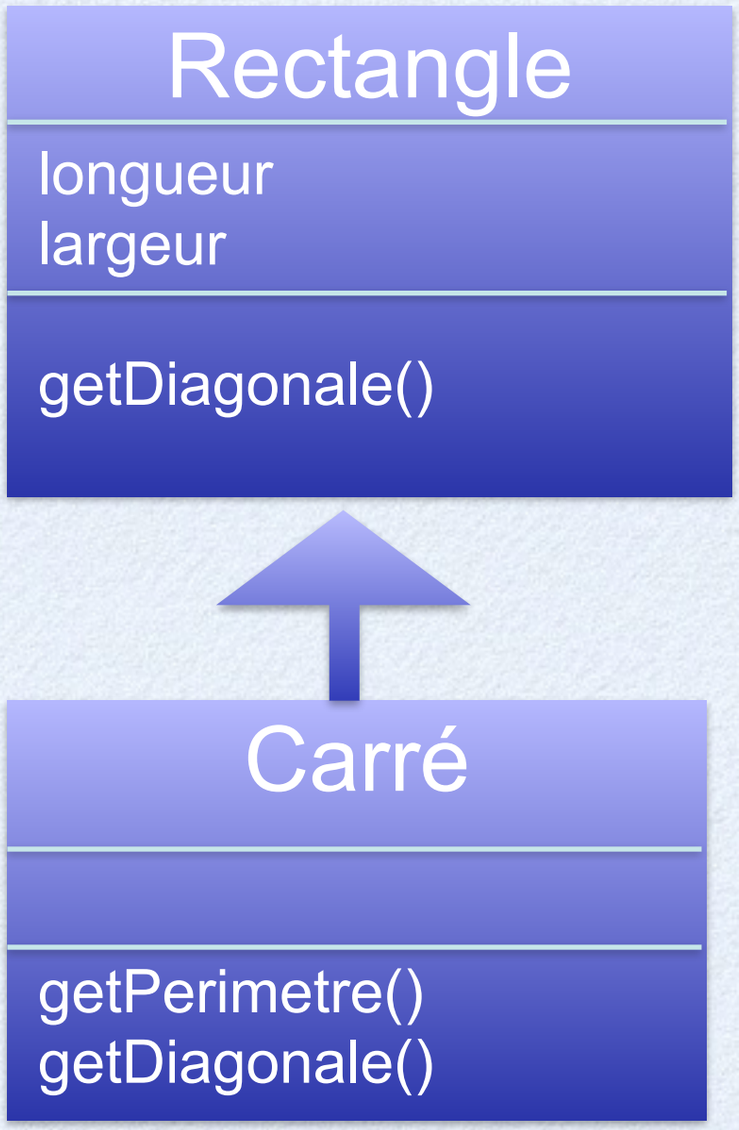
L'Objet A est une instance de la Class Carré:

- L'objet A a pour caractéristiques :
  - Longueur = 5 cm
  - Largeur = 5 cm

- L'objet A a pour méthodes :
  - getPerimeter()
  - getDiagonale()

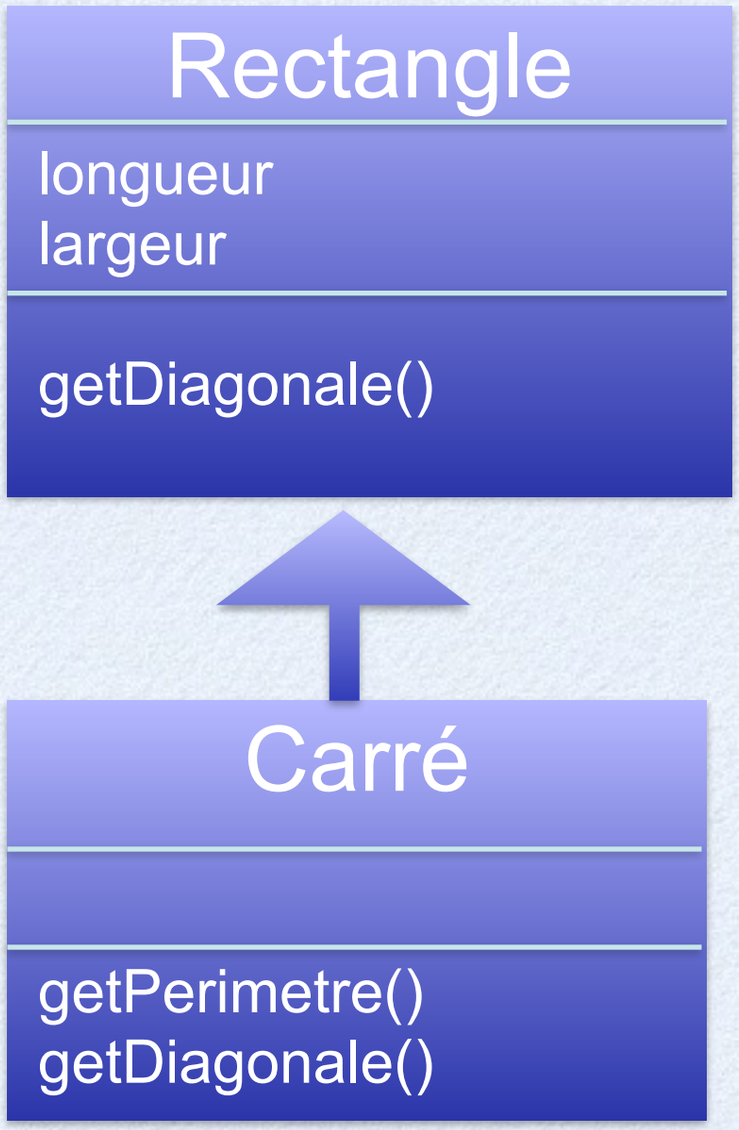
Caractéristiques des POO : Héritage

- L'héritage simple :



Caractéristiques des POO : Héritage

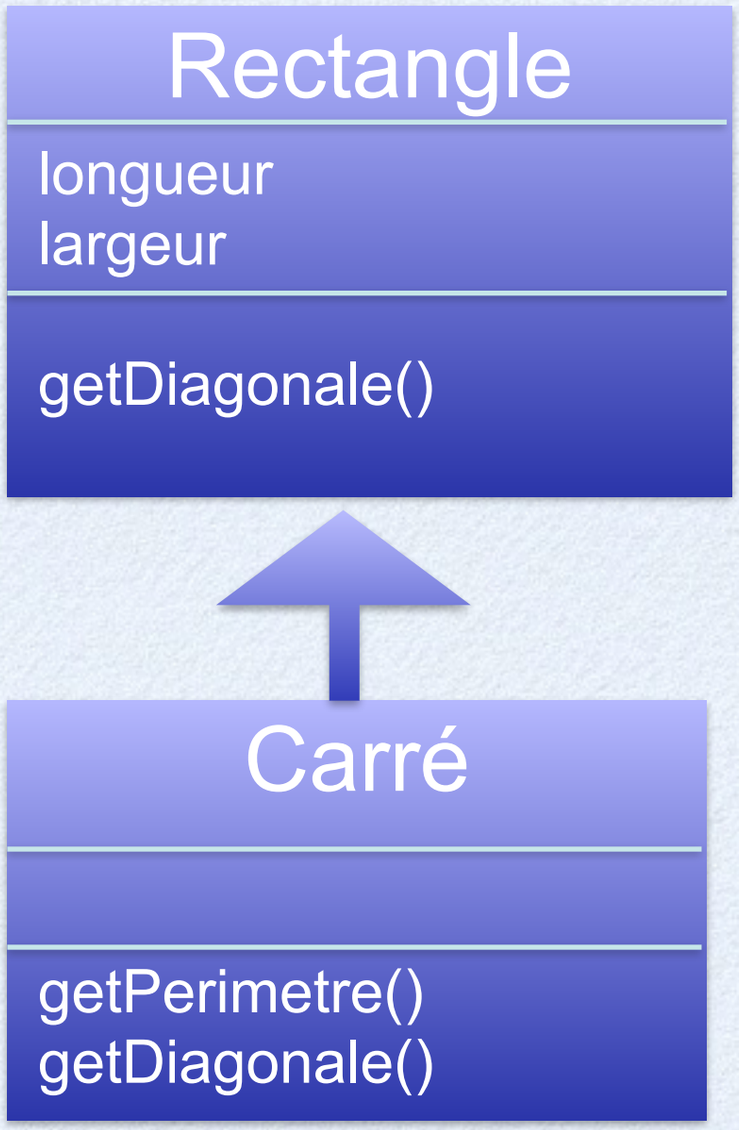
- L'héritage simple :



Instance de carré A

Caractéristiques des POO : Héritage

- L'héritage simple :



L'Objet A est une instance de la Class Carré:

- L'objet A a pour caractéristiques :
  - Longueur = 5 cm
  - Largeur = 5 cm
- L'objet A a pour méthodes :
  - getPerimeter()
  - **getDiagonale() de la class Rectangle?**
  - **getDiagonale() de la class Carre?**

Caractéristiques des POO : Polymorphisme

- **Polymorphisme ad-hoc** : corresponds à définir plusieurs méthodes dans une même classe qui possèdent le même nom mais des paramètres différents.
  - getPerimetre();
  - getPerimetre(int Longueur, int Largeur);

- **Polymorphisme d'inclusion** : est lié à la redéfinition des méthodes. Lors de l'héritage une classe fille peut avoir besoin de modifier le comportement de méthodes de la classe mère. Les méthodes sont alors « sur écrite ».
  - getDiametre(); // de la Classe Rectangle
  - getDiametre(); // de la Classe Carre

- Le compilateur choisi la bonne méthode en fonction des paramètres et de la nature de l'objet

Organisation du cours

I. Introduction à la POO

II. Lien entre le POO et le MOO

III. Caractéristiques des POO

IV. Présentation de Java

V. Projet Java

Java : quelques chiffres

- Quelques chiffres et faits à propos de Java en 2011 :
  - 97% des machines d'entreprises ont une JVM installée
  - Il y a plus de 9 millions de développeurs Java dans le monde
  - Java est un des langages les plus utilisé dans le monde
  - Plus de 3 milliards d'appareils mobiles peuvent mettre en oeuvre Java
  - Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année

Java : en quelques mots

- Java est apparu en 1995.

- Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C.

- Ses caractéristiques ainsi que la richesse de son écosystème et de sa communauté lui ont permis d'être très largement utilisé pour le développement d'applications de types très disparates.

- Java est notamment largement utilisé pour le développement d'applications d'entreprise et mobiles.

Java : Petite et Grande histoire

- Il était une fois une équipe d'ingénieurs de chez Sun dont le but était de développer un langage pour les Assistants Personnels (Smartphones, etc.). Ce qui impliquait un langage :
  - Compact
  - Orienté réseau
- Ce langage s'appelait Oak

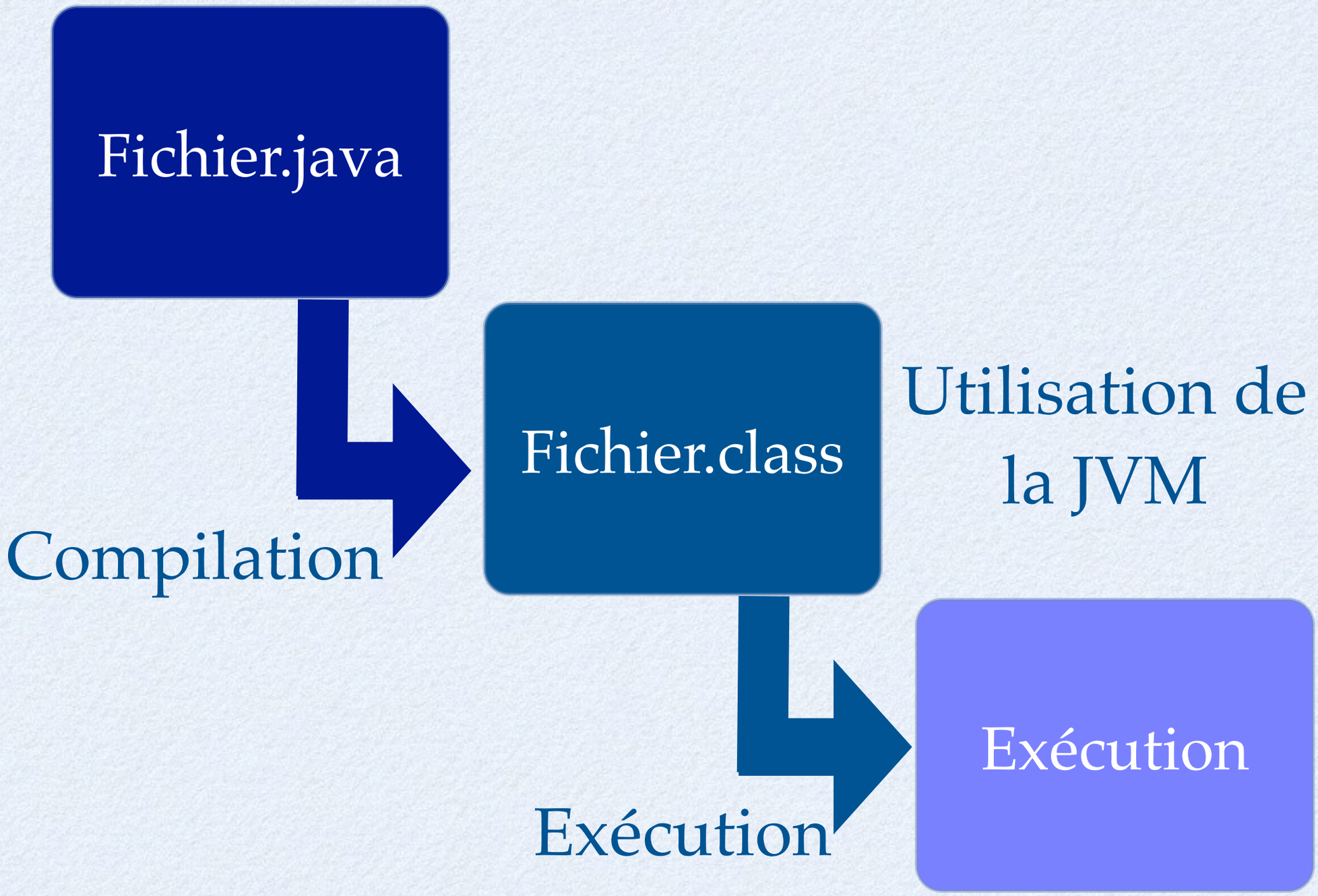
Java : Petite et Grande histoire

- Oak a finalement dérivé puisqu'il s'appliqua aux applications distribuées sur Internet

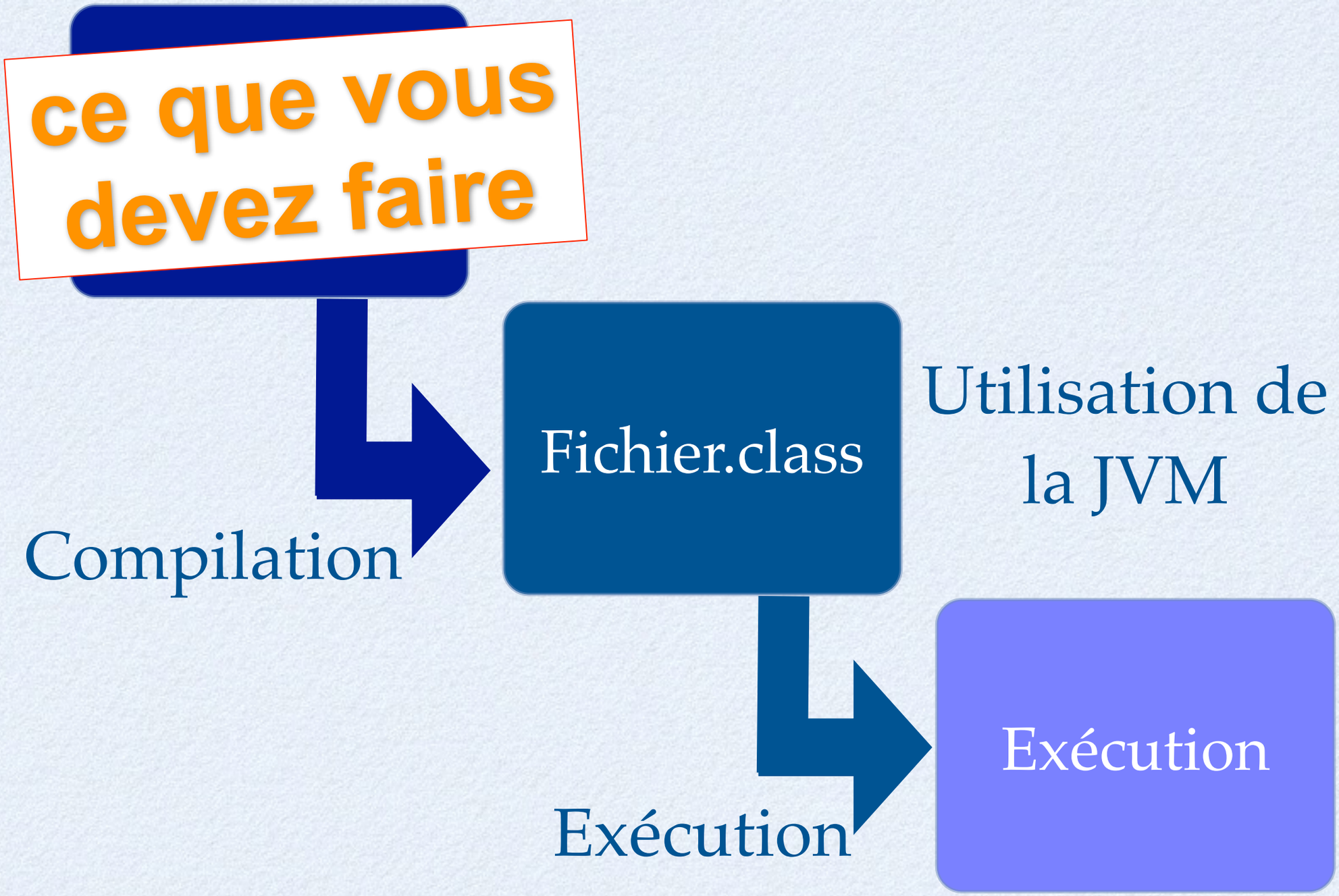
- **JAVA signifie Kawa**

- La légende veut que l'idée de créer ce nouveau langage soit née d'une discussion à bâtons rompus autour d'une tasse de café… Ce qui explique que Sun ait choisi le nom de JAVA en 1995 (Java signifie café en argot américain) pour le lancement des premières versions.

Java : environnement



Java : environnement



Correspondance UML et JAVA

UML

une classe

Cercle
Nom Rayon Couleur
tracer()

Java

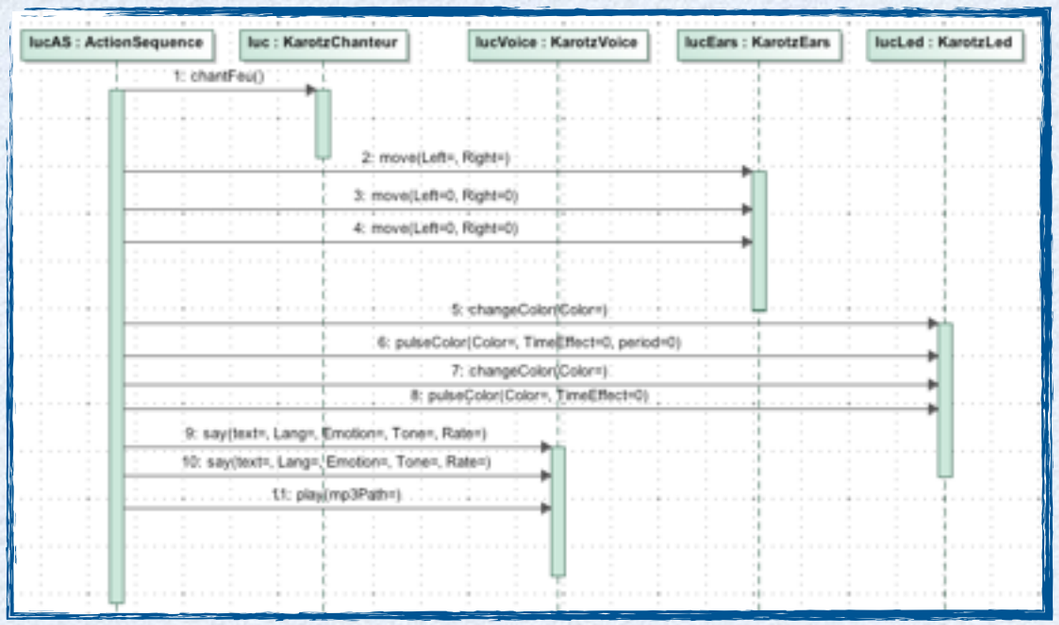
une classe

```java
public class Circle {

    protected String nom;
    private int rayon;
    public String color;


    public Circle() {
        this.nom = "instance";
        this.rayon = 10;
        this.color = "BLUE";
    }

    public int tracer(){
        return 2*rayon;
}
```

Correspondance UML et JAVA

UML

un diagramme de séquence



Java

une méthode d'une classe

```java
public void chanterCalme(){

    this.getEars().move(0, 0, true);
    this.getEars().moveRelative(-10, 22, false);

    this.getLed().changeColor(LedColor.RED);
    this.getLed().fadeColor(LedColor.BLUE, 1000, true);
    this.getLed().pulseColor(LedColor.WHITE, 1000, true);
    this.getLed().pulseColor(LedColor.BLACK, 1000, 500, false);
    this.synchronize(5000);
    this.getLed().changeColor(LedColor.ORANGE);
    KarotzVoice voice = this.getVoice();
    voice.say("Je suis un karote je ne veux pas finir en terrine", VoiceLanguage.FRENCH, true);
    voice.say("Je suis un karote je ne veux pas finir en terrine", VoiceLanguage.FRENCH, VoiceEmotion.ANGRY, tr
    voice.say("Je suis un karote je ne veux pas finir en terrine", VoiceLanguage.FRENCH, VoiceEmotion.HAPPY, Vo
    voice.say("Je suis un karote je ne veux pas finir en terrine", VoiceLanguage.FRENCH, null, VoiceTone.SUPER_
    voice.say("Je suis un karote je ne veux pas finir en terrine", VoiceLanguage.FRENCH, VoiceEmotion.HAPPY, nu
    voice.play("bock.mp3", true);
    //voice.say("Please interupt me", VoiceLanguage.ENGLISH, false);
    this.getLed().pulseColor(LedColor.BLUE, 300, true);
    //voice.stop();

    System.out.println("------------ Simulation ------------");
    this.getActionSequence().simulate();
    System.out.println("------------ Execution ------------");
    this.getActionSequence().execute();
}
```

Les structures de base du langage Java

Création d'une classe Java

```
public class Circle {

    protected String nom;
    private int rayon;
    public String color;


    public Circle() {
        this.nom = "instance"
        this.rayon = 10;
        this.color = "BLUE";
    }

    public int tracer(){
        return 2*rayon;
}
```

Nom de la Classe

Attributs de la Classe

Constructeurs de la Classe

Méthodes de la Classe

Sens d'écriture et de lecture

Création d'une classe Java

```
public class Circle {

    protected String nom;
    private int rayon;
    public String color;


    public Circle() {
        this.nom = "instance"
        this.rayon = 10;
        this.color = "BLUE";
    }

    public int tracer(){
        return 2*rayon;
}
```

Nom de la Classe

Attributs de la Classe

Constructeurs de la Classe

Méthodes de la Classe

Création d'une classe Java

```
public class Circle {

    protected String nom;
    private int rayon;
    public String color;


    public Circle() {
        this.nom = "instance"
        this.rayon = 10;
        this.color = "BLUE";
    }

    public int tracer(){
        return 2*rayon;
}
```

Nom de la Classe

Attributs de la Classe

Constructeurs de la Classe

Méthodes de la Classe

Toutes les attributs et plus généralement toutes les variables doivent être typées

Création d'une classe Java

```
public class Circle {

    protected String nom;
    private int rayon;
    public String color;

    public Circle() {
        this.nom = "instance"
        this.rayon = 10;
        this.color = "BLUE";
    }

    public int tracer(){
        return 2*rayon;
}
```

Nom de la Classe

Attributs de la Classe

Constructeurs de la Classe

Méthodes de la Classe

Toutes les attributs et plus généralement toutes les variables doivent être typées

Résultat de l'encapsulation

Les structures de base du Java

- Une instruction se termine par ;
- Un bloc d'instruction s'écrit entre accolades { . . . }
- Les commentaires :
  - // commentaire sur une ligne
  - /* commentaires sur une ou plusieurs lignes */

Les structures de base : Convention de codage

- Identificateur de classe et d'interface :
  - Première lettre en majuscule
  - Exemple : Classe
- Identificateur de variable :
  - Première lettre en minuscule
  - Exemple : variable
- Constante :
  - Nom en majuscule
  - Exemple : CONSTANTE
- Méthode :
  - Première lettre ne minuscule (sauf pour les constructeurs)
  - Exemple : methode()

Les structures de base : Les opérateurs

Opérateur	Signification
++	Incrémentation de 1 (x++⇔x=x+1)
--	Décrémentation de 1 (x--⇔x=x-1)
+	Addition / signe positif / concaténation
-	Soustraction / signe négatif
!	Opposé logique (!true=false)
(type)	Changement de type pour l'objet
*	multiplication
/	division
%	modulo

Opérateur	Signification
=	assignation
==	Égal à
!=	Différent de
&&	ET logique
||	OU logique
instanceof	Test de type de l'objet
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à

Les structures de base : les méthodes

- Méthode de lancement des programmes:
  - **public static void main (String[ ] args){**

        / / déroulement du programme

    }

- Ecrire sur la console :
  - **System.out.println("**Hello World**");**

    => écrit sur la console : Hello World.

Les structures de base : l'héritage

- L'héritage se réalise grâce au mot clé :

extends

```java
public class Circle extends Forme{

    private String nom;
    private double rayon;
    /**
     *
     */
    public Circle() {
    }
    /**
     * @param nom
     * @param rayon
     */
    public Circle(String nom, double rayon) {
        super();
        this.nom = nom;
        this.rayon = rayon;
    }
```

Les structures de contrôles

- Les structures de contrôles sont composées :
  - D'un ou plusieurs Mot Clé
    - Mot clé permettant au compilateur d'interpréter la structure de contrôle.
    - Les mots clés sont généralement des minuscules.
  - D'une Condition
    - Correspond toujours à un Booléen si vrai le bloc d'instruction est exécuté.
  - D'un ou plusieurs Blocs d'instruction
    - Toujours entre accolades { }

Les structures de contrôles

- Les structures de contrôles sont composées :
  - D'un ou plusieurs Mot Clé
    - Mot clé permettant au compilateur d'interpréter la structure de contrôle.
    - Les mots clés sont généralement des minuscules.
  - D'une Condition
    - Correspond toujours à un Booléen si vrai le bloc d'instruction est exécuté.
  - D'un ou plusieurs Blocs d'instruction
    - Toujours entre accolades { }

```
Mot Clé (Condition) {
        Bloc d'instruction;
}
```

Les structures de contrôles : Si

```
if (Condition) {
    Bloc d'instruction si la condition est vraie;
} else {
   Bloc d'instruction si la condition est fausse;
}
```

Soit A un rectangle et d un entier

SI (diagonale de A = 5)
Alors

d = diagonale de A

Sinon

d = diagonale de A + 2

Fin Si

```
if(A.getDiagonale() == 5)
{

   d = A.getDiagonale();

} else {

  d = A.getDiagonale() -2;
}
```

Les structures de contrôles : les boucles

Les structures de contrôles : les boucles

- Rappels sur les boucles :
  - **Pour … Faire**
    - On l'utilise quand : **ON CONNAÎT LE NOMBRE D'ITERATION**

Les structures de contrôles : les boucles

- Rappels sur les boucles :
  - **Pour … Faire**
    - On l'utilise quand : **ON CONNAÎT LE NOMBRE D'ITERATION**
  - **Tant que … Faire**
    - On l'utilise quand : **ON NE CONNAÎT PAS LE NOMBRE D'ITERATION**
    - La condition d'arrêt est testée avant l'exécution du bloc d'instruction

Les structures de contrôles : les boucles

- Rappels sur les boucles :
  - **Pour … Faire**
    - On l'utilise quand : **ON CONNAÎT LE NOMBRE D'ITERATION**
  - **Tant que … Faire**
    - On l'utilise quand : **ON NE CONNAÎT PAS LE NOMBRE D'ITERATION**
    - La condition d'arrêt est testée avant l'exécution du bloc d'instruction
  - **Faire … Jusqu'à**
    - On l'utilise quand : **ON NE CONNAÎT PAS LE NOMBRE D'ITERATION**
    - La condition d'arrêt est testée après l'exécution du bloc d'instruction

Les structures de contrôles : Pour

```
for (Condition1; Condition2; Condition3) {
    Bloc d'instruction si la condition est
vraie;
}
```

Condition1 : valeur initiale de la variable d'itération
Condition2 : condition de fin de la boucle
Condition3 : définition du pas d'incrémentation

Pour (i allant de 0 à 10) Faire

d = d + i

Fin Pour

```
for(int i=0; i<=10; i++) {

   d = d+i;
}
```

Les structures de contrôles : Tant que

```
while (Condition) {
    Bloc d'instruction si la condition est
vraie;
}
```

Tant que (i est différent de 10) Faire

i=i+1

Fin Pour

```
While(i !=10) {

    i++;

}
```

Les structures de contrôles : Tant que

```
do {
    Bloc d'instruction si la condition est
vraie;
} while (Condition)
```

Répéter

i=i+1

Jusqu'à (i est différent de 10)

```
do {

   i++;

} While(i !=10)
```

Organisation du cours

I. Introduction à la POO

II. Lien entre le POO et le MOO

III. Caractéristiques des POO

IV. Présentation de Java

V. Projet Java

Projet Java

- Présentation de l'environnement de développement : ECLIPSE
- Explication pour l'utilisation du Karotz
- Explication pour l'utilisation du MusiqueManager
- Explication pour l'utilisation du AnimationManager