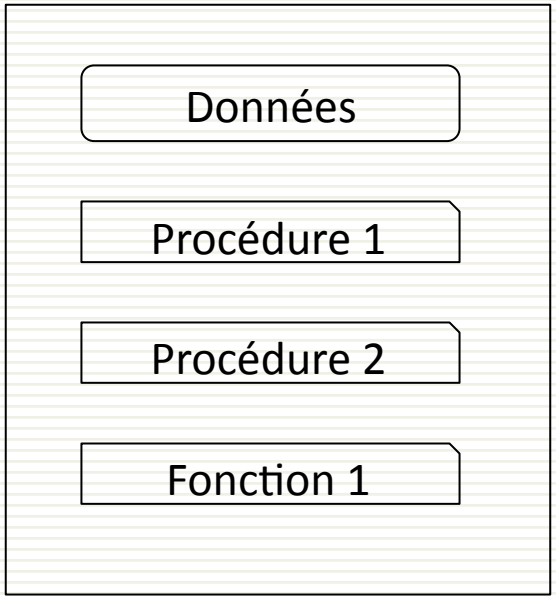
IUT 2 Grenoble Université Pierre-Mendès France Informatique

EF1
Bases de la Programmation Orientée Objet

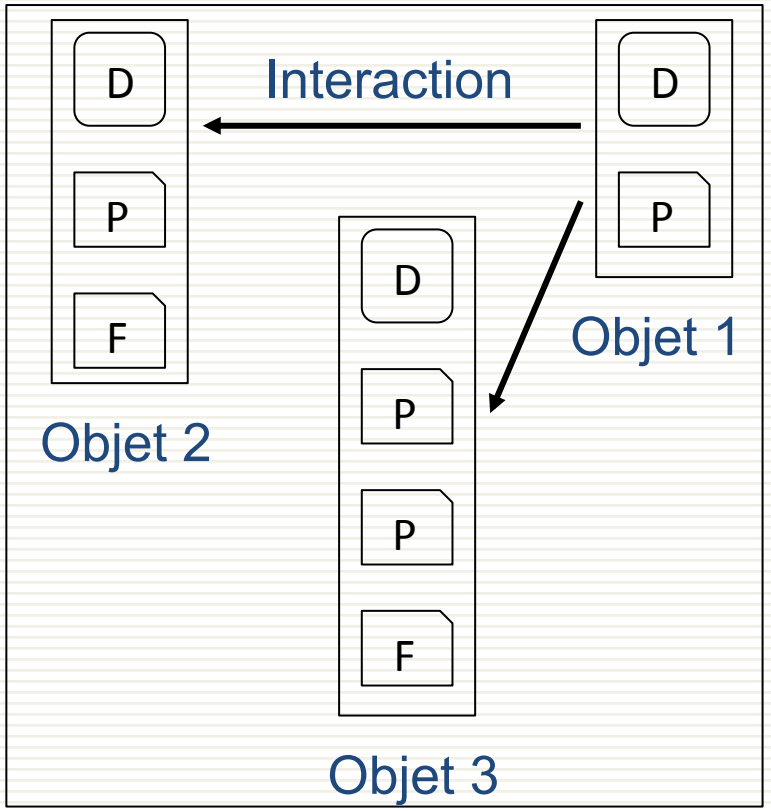
- Introduction à la programmation orientée objet
  - Premier objet, notion de classe, Encapsulation

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET

Programmation orientée objet (1/2)



Programmation classique
Séparation entre
- procédures et fonctions
- données



Programmation orienté objet
Programme = ensemble d'objets
Objet = données + méthodes (proc & fct)
Interaction = par envoie de messages

Programmation orientée objet (2/2)

- ✓ Objectifs de la POO
  - Programmer par « composants »
  - Améliorer la conception et la maintenance
  - Faciliter la réutilisation du code
  - Faciliter l'évolution du code (nouvelles fonctionnalités)

- ✓ Apports de la POO
  - Objet, Classe, Encapsulation, Héritage, Polymorphisme, etc.

Premier objet : String (1/6)

- La classe String permet de créer des objets de type chaîne de caractères
- `char data[] = {'a', 'b', 'c'};`

```
//variable de type tableau de caractères initialisée
String s1 = new String(data);
   //variable, objet, de type String construite (new)
   //pour contenir la chaîne "abc"
   //s1 est une instance d'objet
String s2 = "abc"; //Simplification d'écriture
```

- Documentation :
  http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Premier objet : String (2/6)

✓ Extrait de la documentation

- Constructors

CONSTRUCTEURS D'UN OBJET STRING

  - **`String()`**
    - Initializes a newly created String object so that it represents an empty character sequence.
  - **`String(char[] value)`**
    - Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

- Methods

MÉTHODES D'UN OBJET STRING

  - **`Char charAt(int index)`**
    - Returns the char value at the specified index.
  - **`Boolean equals(Object anObject)`**
    - Compares this string to the specified object.
  - **`int length()`**
    - Returns the length of this string.

Premier objet : String (3/6)

s1 & s2 sont 2 instances d'objets de la classe String valant "abc"

- ✓ On dispose de la méthode :
  - **`Boolean equals(Object anObject)`**
    - Compares this string to the specified object.
- ✓ Exemple d'usage :
  - **`s1.equals(s2)`** **`// retourne un booléen`**
- ✓ Explication, lecture
  - appliquer la méthode **`equals()`** à l'objet **`s1`**
  - l'objet **`s1`** est cette String, cet objet (this string)
  - l'objet **`s2`** est l'objet spécifié (the specified object)

source Java

```java
public class JavaStringTest {
  public static void main(String[] args) {
    // variable de type tableau de caractères initialisée
    char data[] = {'a', 'b', 'c'};
    // variable, objet, de type String construite (new)
    // pour contenir la chaîne "abc"
    // s1 est une instance d'objet
    String s1 = new String(data);
    // variables objet de type string construites implicitement
    // avec "abc" et "def" : simplification d'écriture
    String s2 = "abc";
    String s3 = "def";
    System.out.println("La chaîne s1 vaut : " + s1);
    System.out.println("La chaîne s2 vaut : " + s2);
    System.out.println("La chaîne s3 vaut : " + s3);
    System.out.println("s1.equals(s2) retourne : " + s1.equals(s2) + " (true attendu)");
    System.out.println("s2.equals(s3) retourne : " + s2.equals(s3) + " (false attendu )");
    }
}
```

Premier objet : String (5/6)

s1 & s2 sont 2 instances d'objets de la classe String valant "abc" & "def"

- ✓ On dispose de la méthode :
  - **`String concat(String str)`**
    - Concatenates the specified string to the end of this string
- ✓ Exemple d'usage :
  - **`String s3 = s1.concat(s2)`**
- ✓ Explication, lecture
  - appliquer la méthode **`concat()`** à l'objet **`s1`**
  - l'objet **`s1`** est cette String, cet objet (this string)
  - l'objet **`s2`** est l'objet spécifié (the specified object)

source Java

```java
public class StringConcatTest {

  public static void main(String[] args) {

    String s1 = "abc";
    String s2 = "def";

    // concaténer s2 à s1
    // méthode concat() appliquée à l'objet s1
    String s3 = s1.concat(s2);
    System.out.println("s1 : " + s1 + " (\"abc\" attendu)");
    System.out.println("s2 : " + s2 + " (\"def\" attendu)");
    System.out.println("s3 : " + s3 + " (\"abcdef\" attendu)");
  }

}
```

trace

```
s1 : abc ("abc" attendu)
s2 : def ("def" attendu)
s3 : abcdef ("abcdef" attendu)
```

Notion de classe

- ✓ Une **classe** est un type décrivant un ensemble d'objets ayant la même structure de données (**attributs**) et le même comportement (**méthodes**)

- ✓ La notion de **classe** est une généralisation de la notion de type déjà rencontrée dans les langages impératifs
  - elle ajoute le comportement (méthodes) à la structure de données (attributs)

- ✓ Un **objet** est une **instance** (une réalisation) d'une classe à laquelle il appartient
  - il peut être vu comme une variable initialisée dotée de méthodes (son comportement)

Notion de classe

- ✓ La classe Etudiant
  - données

nom : String	prenom : String
login : String	notes : …

  - méthodes

getNom() : String setNom(String)	getPrenom() : String setPrenom(Sting)
getLogin() : String setLogin(String)	setNotes(…) getMoyenne() : double

Spécification d'une classe (exemple)

Diagramme de classe UML

- ✓ La classe Etudiant

Etudiant
– login : String – nom : String – prénom : String – notes[0..*] : int
+ Etudiant(String login) + Etudiant(String nom, String prénom, String login) + getLogin() : String + getNom() : String + getPrenom() : String + getMoyenne() : double + getNomComplet() : String + getMail() : String + setLogin(String login) + setNom(String nom) + setPrenom(String prénom) + setNotes() + affiche()

Légende

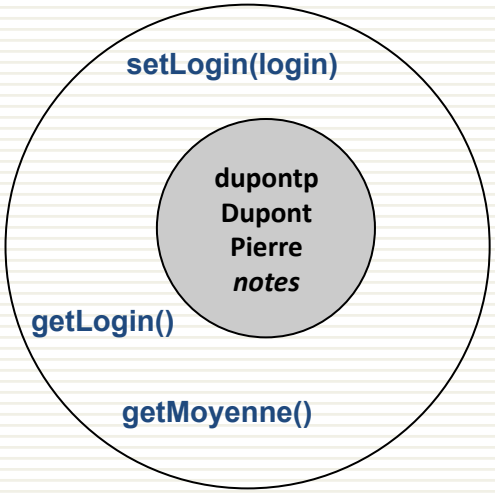
- ✓ 3 zones

Nom de classe
définition attributs
définition méthodes

- ✓ préfixe –
  - indique un membre privé
- ✓ préfixe +
  - indique un membre public
- ✓ préfixe absent
  - indique un constructeur

membre : attribut ou méthode

Notion d'objet



- ✓ Un objet est défini par
  - Un état
    - Représenté (caractérisé) par des données (**attributs**)
  - Un comportement
    - Défini par des procédures et fonctions (**méthodes**) qui modifient les données et envoient des messages à d'autres objets
  - Une identité (unEtudiant)
    - Permet de distinguer un objet d'un autre objet

Utilisation d'une classe (1/2)

- la classe Etudiant définie dans le paquetage Etudiant
- la classe main() définie dans le même paquetage sera par exemple :

TestEtudiant.java

```
package Etudiant;
public class TestEtudiant {
  public static void main(String[] args) {
    // Déclaration et construction de deux objets
    Etudiant et1 = new Etudiant("doej");
    Etudiant et2 = new Etudiant("martinf", "martin", "francis");

    // Affichage des objets de type Etudiant
    System.out.println("\nAfficher les deux étudiants créés :");
    et1.affiche();
    et2.affiche();
  }
}
```

Utilisation d'une classe (2/2)

TestEtudiant.java

```
package Etudiant;
public class TestEtudiant {
  public static void main(String[] args) {
    // Déclaration et construction de deux objets
    Etudiant et1 = new Etudiant("doej");
    Etudiant et2 = new Etudiant("martinf", "martin", "francis");

    et1.setLogin("bruneth");          // Modifie le login de l'instance et1
    et1.setNom("brunet");             // Modifie le nom de l'instance et1
    et1.setPrenom("hervé");           // Modifie le prénom de l'instance et1

    String login = et2.getLogin();    // accède au login de l'instance et2
    String prenom = et2.getPrenom(); // accède au prénom de l'instance et2
    String nom = et2.getNom();        // accède au nom de l'instance et2

    // Affichage de l'état de l'objet et1
    et1.affiche();
    // Affichage des informations récupérées sur l'objet et2
    System.out.println("Etudiant de login" + login + "nom complet : "+ prenom + nom);
  }
}
```

Principe d'encapsulation

Un utilisateur extérieur ne doit pas modifier directement les données et risquer de mettre en péril l'état et le comportement de l'objet

- ✓ Comment ?
  - Protéger les données contenues dans un objet
  - Proposer des méthodes pour manipuler les données d'un objet

Principe d'encapsulation

- ✓ Comment protéger les données contenues dans un objet
  - les attributs seront privés, c'est-à-dire consultables ou modifiables uniquement par des méthodes de la classe de l'objet
- ✓ Comment proposer des méthodes pour manipuler (consulter, modifier) les données d'un objet
  - les méthodes de la classe de l'objet utilisables par un utilisateur de la classe seront publiques
  - **Conséquence** : la classe devra fournir des méthodes publiques pour consulter (*accesseurs* ou *getters*) et/ou modifier (*mutateurs* ou *setters*) les attributs quand c'est nécessaire

Principe d'encapsulation

- ✓ Cohérence des données d'un objet ?
  - les méthodes qui modifient des attributs doivent garantir la cohérence de l'objet (la cohérence des valeurs données aux attributs)

- ✓ Les attributs d'une classe sont tous privés !
- ✓ Les méthodes d'une classe sont-elles toutes publiques ?
  - sont **publiques** les méthodes utilisables par un utilisateur de la classe (ce sont elles qu'il doit utiliser)
  - sont **privées** les méthodes de « service » aux méthodes publiques non utilisables par un utilisateur de la classe (il ne peut, ne doit, pas les utiliser)

Principe d'encapsulation

- ✓ Privé vs Publique
  - Les membres privés (**`private`**) ne sont visibles qu'à l'intérieur de la classe (par les méthodes de la classe donc)
  - Les membres publique (**`public`**) sont visibles par toutes les parties de programmes (par l'utilisateur de la classe ou par d'autres classes par exemple)

- ✓ ***Note** : le terme « membre » désigne un attribut ou une méthode.*

Exemple d'encapsulation (1/2)

```
public class Etudiant {

   /* Attributs */
   private String login;
   private String nom;
   private String prenom;
   private int notes[];

   /* Constructeurs */
   public Etudiant(String login) {
          this.login = login;
          nom = "doe";
          prenom = "john";
   }
```

…

Le mot clé `this` désigne l'objet dans lequel on se trouve !
(peut être facultatif si pas d'ambiguïté)

```
/* Méthodes */
public String getLogin() {
      return login;
}

public String getNom() {
      return nom;
}
```

ACCESSEURS/GETTERS

```
public void setLogin(String login) {
      this.login = login;
}

public void setNom(String nom) {
      this.nom = nom;
}
```

MUTATEURS/SETTERS

…

Exemple d'encapsulation (2/2)

```
public class Etudiant {

   /* Attributs */
   private String login;
   private String nom;
   private String prenom;
   private int[] notes;

   /* Constructeurs */
   public Etudiant(String login) {
          setLogin(login);
          nom = "doe";
          prenom = "john";
   }

…
```

```
/* Méthodes */
public String getLogin() {
      return login;
}

public String getNom() {
      return nom;
}
```

ACCESSEURS/GETTERS

```
public void setLogin(String login) {
      this.login = login.toLowerCase();
}

public void setNom(String nom) {
      this.nom = nom;
}
```

MUTATEURS/SETTERS

…

Une méthode pour garantir la cohérence de l'attribut login

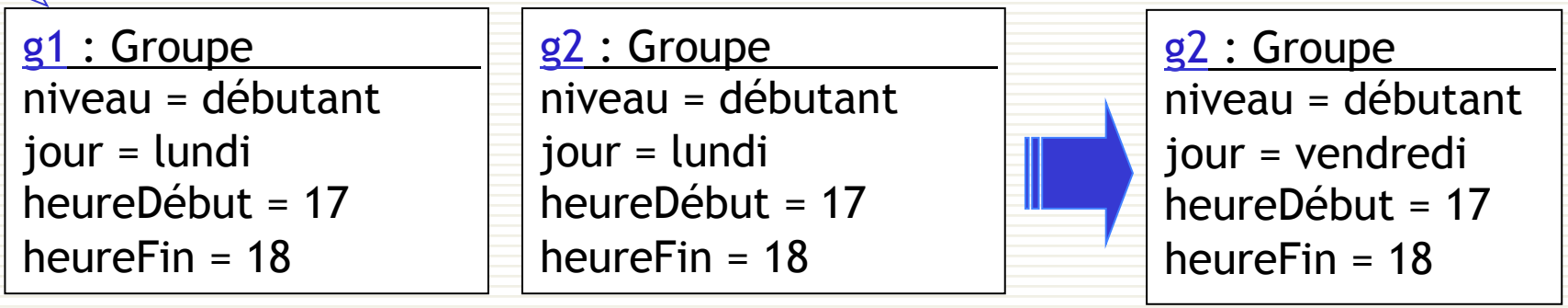
Avantages de l'encapsulation

- ✓ Avantage majeur
  - **Protéger** les données
- ✓ Autres avantages
  - Améliorer la **conception** et la **maintenance**
  - Faciliter la **réutilisation** du code
  - Faciliter d'**évolution** du code
    - nouvelles fonctionnalités

Identité d'un objet (1/2)

- ✓ Tout objet est identifié par un nom (identificateur) externe indépendant de son état.
- ✓ Le nom est unique et n'est pas défini par son contenu.
  - permet de faire référence à un objet.
  - permet de distinguer des objets ayant mêmes valeurs d'attributs.

nom de l'objet



Dans une école de natation : 2 groupes de natation **différents** de même niveau ayant leur séance au même moment.

Les objets g1 et g2 sont des « jumeaux »

Après un changement de jour pour le groupe g2.

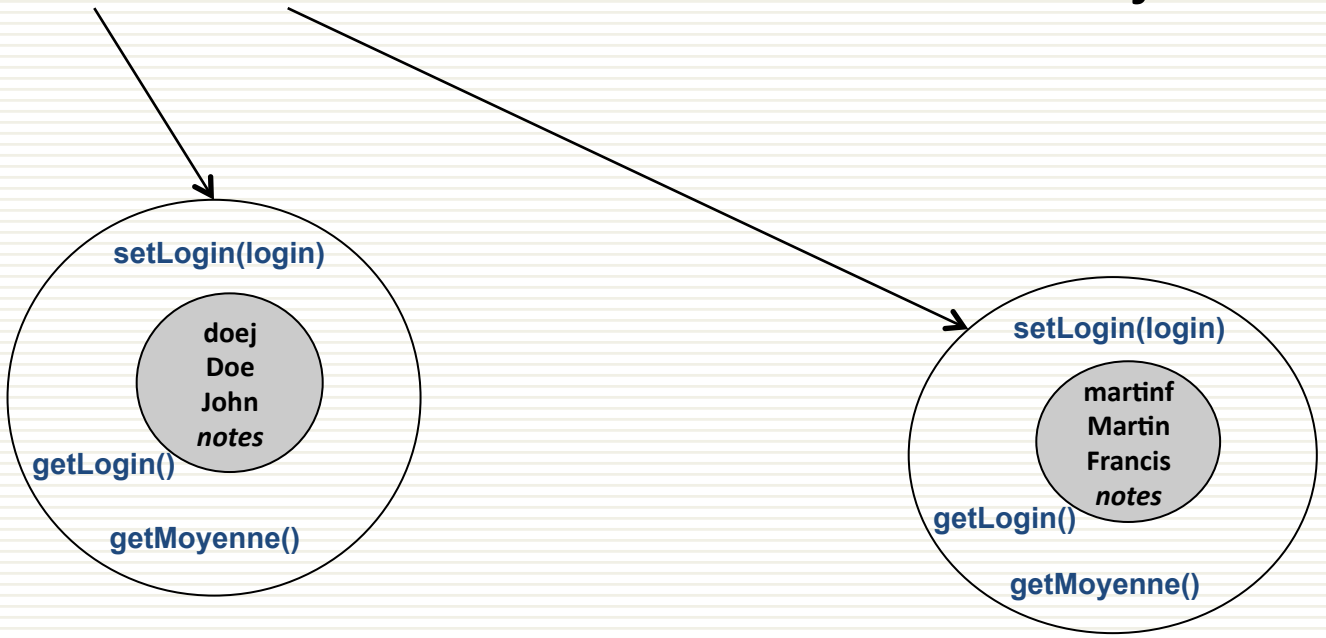
L'objet g2 a changé d'état.

Identité d'un objet (2/2)

- ✓ Un objet est une instance de classe

```
Etudiant et1 = new Etudiant("doej");
Etudiant et2 = new Etudiant("martinf", "martin",
   "francis");
```

- ✓ et1 et et2 sont des références à l'objet

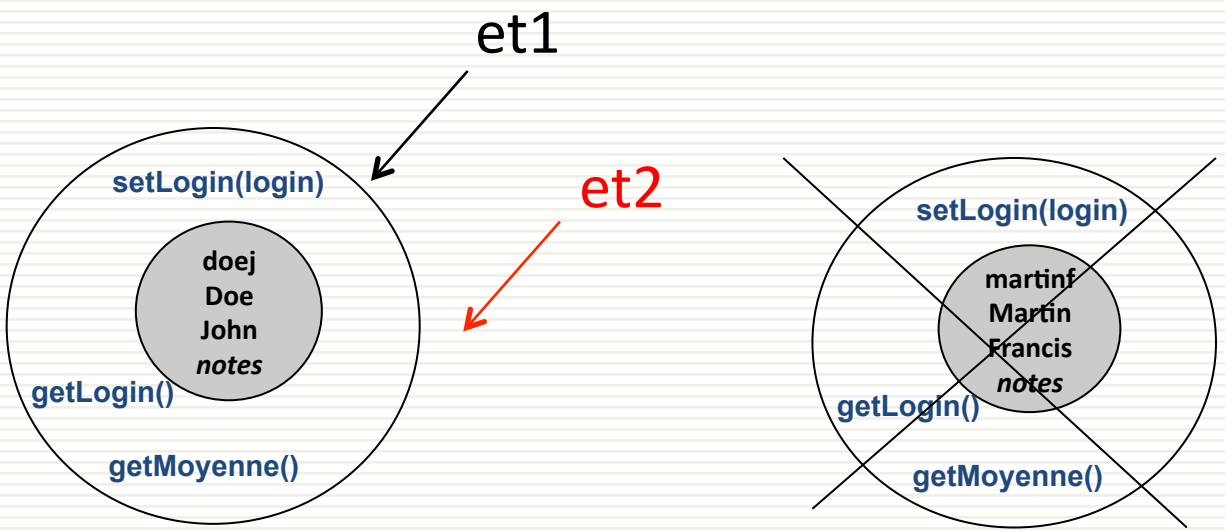


Identité d'un objet (2/2)

- ✓ Un objet est une instance de classe

```
Etudiant et1 = new Etudiant("doej");
Etudiant et2 = new Etudiant("martinf", "martin", "francis");
```

- ✓ Effet de l'instruction : et2 = et1 ;



Exercice 3 : Figure2D classe Cercle V1

- ✓ En respectant le principe d'encapsulation
  - créer une classe **`Cercle`**
    - centre du cercle (`x`, `y`)
      - `x` : abscisse du centre
      - `y` : ordonnée du centre
    - périmètre : 2 × π × `rayon`
    - surface : π × `rayon`
    - `Cercle(int monRayon)`
      - cercle de centre (0, 0)
  - tester la classe cercle

Cercle
– x : int – y : int – rayon : int
+ Cercle(int monRayon) + Cercle(int xOrigine, int yOrigine, int monRayon) + setRayon(int nouveauRayon) + getRayon() : int + getPerimetre() : double + getSurface() : double + deplaceCentre(int dx, int dy) + affiche()

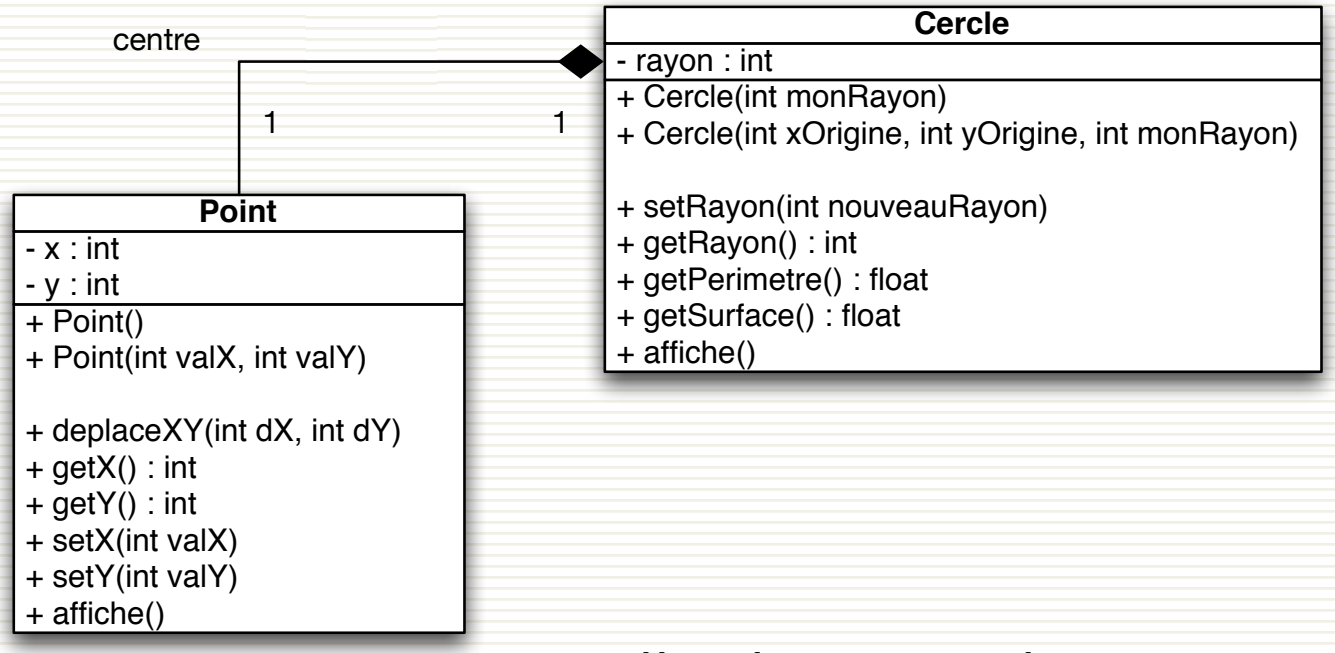
Exercice 4 : Figure2D classe Point

- ✓ On décide maintenant de créer une nouvelle classe **`Point`** qui permettra de définir l'origine de différentes figures en deux dimensions
- ✓ En respectant le principe d'encapsulation
  - créer & tester la classe **`Point`**

Point
- x : int - y : int
+ Point() + Point(int valX, int valY) + deplaceXY(int dX, int dY) + getX() : int + getY() : int + setX(int valX) + setY(int valY) + affiche()

Exercice 5 : Figure2D classe Cercle V2

- ✓ On décide qu'un **`Cercle`** à 2 attributs :
  - un `centre` de type **`Point`** (composition)
  - un `rayon`
- ✓ la représentation UML est la suivante :



  - créer et tester cette nouvelle classe cercle