

R2-01-03

DÉVELOPPEMENT ORIENTÉ OBJETS

QUALITÉ DE DÉVELOPPEMENT

Semaine 1

- Classes, objets, association
- Documentations, commentaires, débogueur

Francis Brunet-Manquat

Université Grenoble Alpes

IUT 2 – Département Informatique

R2-01-03



Deux ressources



R2-01 Développement orienté objets



R2-03 Qualité de développement



Décomposé en



R2-01a : outils et programmation orientée objets POO



Vous allez **utiliser**



R2-01b : modélisation orientée objets







Vous allez **créer**

Déroulement de R2.01a

Semaine type

-  1 TD de 2h : introduction des notions puis debut du TP
-  2 TPs de 2h : fin du TP

Semaine 1 : rappel de R1.01

-  Exercice 1 : la classe String et son utilisation
-  Exercice 2 : création d'une classe
-  Exercice 3 : association entre classes
-  Exercice fil-rouge : un jeu


Evaluations

 Les évaluations seront coordonnées entre R2.01a et R2.01b

 2 examens machines

 Un examen orienté *qualité de développement*

 Un examen orienté *développement orienté objet*





 Contrôle continu : évaluation de vos tps

 Modalités en cours de réflexion





 **POSEZ DES QUESTIONS**

Programmation orientée objets

Objectifs de la POO

-  Améliorer la conception, l'exploitation & la maintenance
-  Programmer par « composants »
-  Faciliter la réutilisation du code
-  Faciliter l'évolution du code (nouvelles fonctionnalités)

Apports de la POO

-  Objet et Classe
-  Encapsulation
-  Héritage
-  Polymorphisme, ...


Exercice 1 : classe String

 Utilisation de méthodes appropriées


 Utilisation de la classe Scanner

 Ecriture de commentaire






 Lecture de documentation Java

 <https://docs.oracle.com/javase/9/docs/api/java/util/Scanner.html>

 <https://docs.oracle.com/javase/9/docs/api/java/lang/String.html>

 <https://docs.oracle.com/javase/9/docs/api/java/lang/StringBuilder.html>

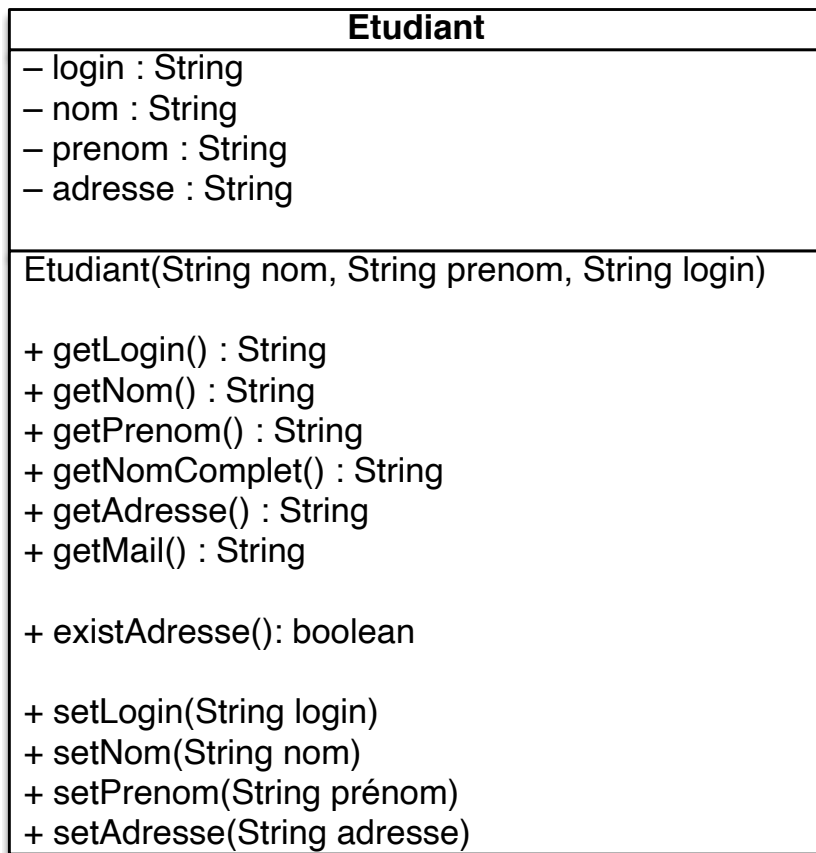
Exercice 2 : notion de classe

-  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)

Exercice 2 : classe Etudiant

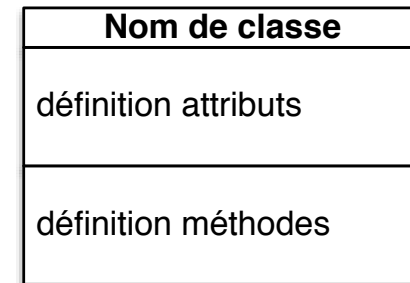
Diagramme de classe UML

La classe Etudiant



Légende

3 zones



préfixe –

On préfère le terme opération
en UML à méthode

 indique un membre privé

préfixe +

 indique un membre public

préfixe absent

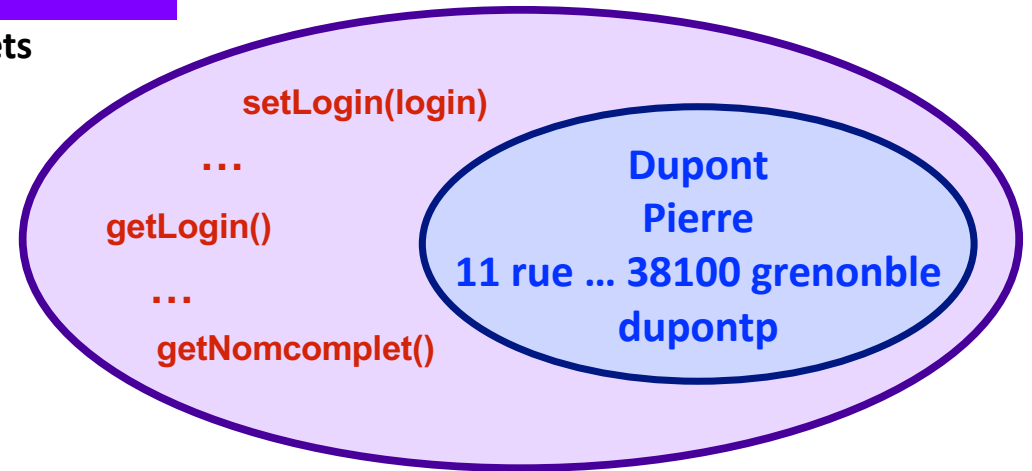
 indique un constructeur

membre : attribut ou méthode

Exercice 2 : notion d'objet


Une **classe** est un type décrivant un ensemble d'**objets**

unEtudiant




 Un objet est défini par

 Un **ÉTAT**

 Représenté (caractérisé) par des **attributs** (données)

 Un **COMPORTEMENT**

 Défini par des **méthodes** (procédures et fonctions) qui modifient les attributs et envoient des messages à d'autres objets

 Une **IDENTITÉ** (unEtudiant)

 Permet de distinguer un objet d'un autre objet

Comment créer un objet ?

```
Etudiant etudiant1 = new Etudiant("dupontp", "pierre", "dupont");
```

Exercice 2 : principe d'encapsulation (1/3)


*Un utilisateur/objet extérieur ne doit pas modifier directement l'**ÉTAT** d'un objet (les données) et risquer de mettre en péril l'**ÉTAT** et le **COMPORTEMENT** de cet objet !*

Pourquoi ? Si une contrainte sur l'état

 Par exemple, un étudiant doit avoir un login en minuscule


Comment ?

 **Protéger les données** contenues dans un objet


 **Proposer des méthodes pour manipuler les données** d'un objet qui assurent la validité/cohérence des données


Exercice 2 : principe d'encapsulation (2/3)

 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 en **assurant validité/cohérence des données**

Exercice 2 : principe d'encapsulation (3/3)



Cohérence des données d'un objet ?



les méthodes qui modifient des attributs doivent garantir la **validité/cohérence de l'objet** (la cohérence des valeurs données aux attributs)



Les **ATTRIBUTS** d'une classe doivent être **PRIVÉS** !



Les méthodes d'une classe sont-elles toutes publiques ?



sont **PUBLIQUES** les **MÉTHODES UTILISABLES PAR UN UTILISATEUR (DÉVELOPPEUR)** de la classe (*il a le droit de les utiliser*)



sont **PRIVÉES** les **MÉTHODES DE « SERVICE »** qui servent aux méthodes de la classe. Elles sont **NON UTILISABLES PAR UN UTILISATEUR** de la classe (*il n'a pas le droit de les utiliser*)

Exercice 2 : encapsulation mise en pratique

```
public class Etudiant {  
  
    private String login;  
    ...  
  
    public Etudiant(String login, String prenom, String nom) {  
        setLogin(login);  
        ...  
    }  
  
    public String getLogin() {  
        return login;  
    }  
  
    public void setLogin(String login) {  
        this.login = login.toLowerCase();  
    }  
}
```

IMPORTANT le mot clé **this** est une variable de référence. Elle pointe vers l'objet courant de la classe. Elle sera le plus souvent utilisée dans une classe pour désambiguïser l'attribut d'une classe d'un paramètre de méthode.

Exercice 2 : tester sa classe

```
public class TestEtudiant {  
  
    public static void main(String[] args) {  
  
        // Création des étudiants  
        // IMPORTANT mettre des valeurs qui ne respectent pas les contraintes  
        // pour vérifier leur prise en compte  
        Etudiant etudiant1 = new Etudiant("DUPONTP", "pierre", "dupoNT");  
        Etudiant etudiant2 = new Etudiant("martinf", "francis", "martin");  
  
        // Ajouter une adresse  
        etudiant2.setAdresse("2 Place Doyen Gosse");  
  
        // Afficher les étudiants  
        EtudiantUtilitaire.afficheEtudiant(etudiant1);  
        EtudiantUtilitaire.afficheEtudiant(etudiant2);  
    }  
}
```


créer

modifier

afficher

- Dans une méthode main
- En utilisant la classe Utilitaire pour afficher

Exercice 3 : associations spécifiques


 Apprendre à mettre en place des associations entre classes spécifiques

 Agrégation

 Composition

 Associer la classe Point à des formes pour représenter :

 Le centre d'un cercle

 L'origine d'un rectangle

Point
- x : int = 0 - y : int = 0
Point() Point(int valX, int valY) + deplaceXY(int dX, int dY) + getX() : int + getY() : int + setX(int valX) + setY(int valY)

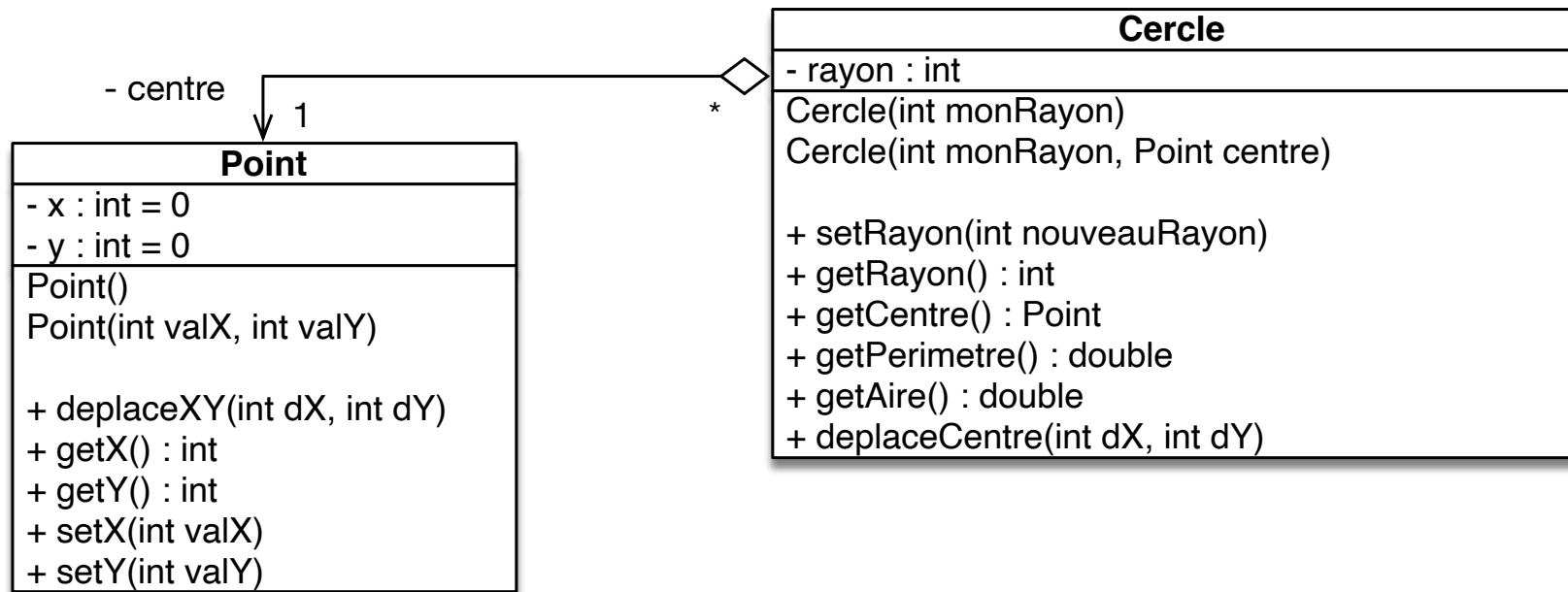
Exercice 3 : agrégation



Association non symétrique, qui exprime un couplage fort et une relation de subordination



un **Cercle** a un attribut de type **Point** pour représenter son centre.



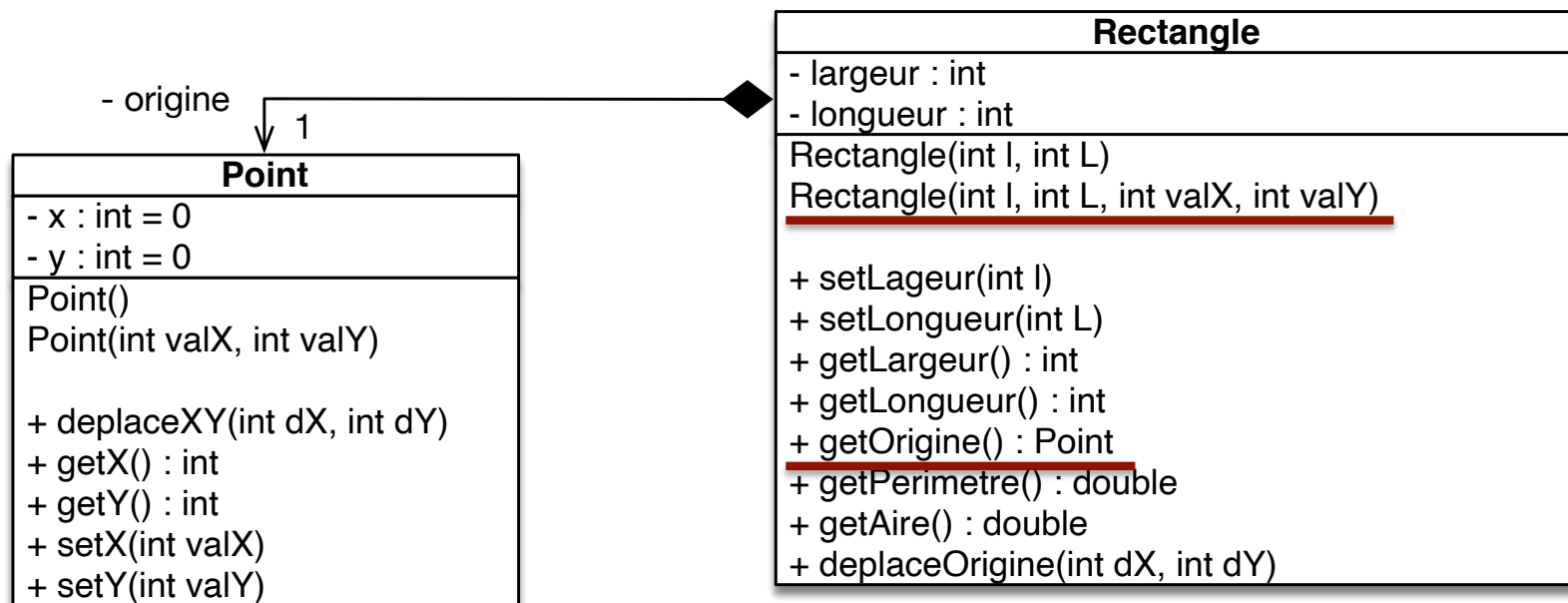
Exercice 3 : composition



Agrégation forte



un **Rectangle** a pour origine un **Point**, a donc un attribut de type **Point** pour représenter son origine **MAIS l'usage du point devra être « caché » et « protégé » des autres classes.**



Exercice fil-rouge



Jeu sur le terminal



Plusieurs étapes tout au long du module



Objectifs : revenir sur les notions abordées dans les TDs et les TPs



Les grandes étapes :



Création de guerriers, de chateaux, d'un plateau de jeu, algorithmique pour les déplacements, pour les combats, etc.