Les classes et les objets en Java

L. Nerima
Université de Genève

Références

- * Conception objet en Java avec BlueJ, David Barnes et Michael Kölling, 4e édition, Pearson Education, 2009

  Version anglaise: Objects First with Java, A Practical Introduction using BlueJ, 6th Edition, Pearson, 2016

- * Bersini, Hugues, La programmation orientée objet, 7e édition, Eyrolles, 2017.

- JAVA: de l'esprit à la méthode, Michel Bonjour, Gilles Falquet, Jacques Guyot et André Legrand, 2e édition, Vuibert, 1999 (Chapitre 6). PDF disponible sur le site Web du séminaire Java.

- Programmer en Java, Claude Delannoy, 10e édition, Eyrolles, 2017.

** Idées de lecture pour la semaine de lecture !*

Plan

- La modélisation orientée objet
- Les classes en Java (rappels sur les classes)
- Classe et instances (objets)
- Déclaration de classes en Java
- Exemples de classe: rectangle, cercle, …
- Les classes dans l'environnement BlueJ
- Création d'instances
- Invocation de méthodes sur un objet
- Assignation et compatibilté des types
- Sous-classes
- La référence

La modélisation orientée objet

Approche centrée sur les objets et les classes d'objets (regroupement d'objets de même type).

Tout est objet: le système informatique sera constitué (uniquement) d'objets. Les objets communiquent entre eux et avec l'extérieur (p.e. interface utilisateur)

Les objets sont:

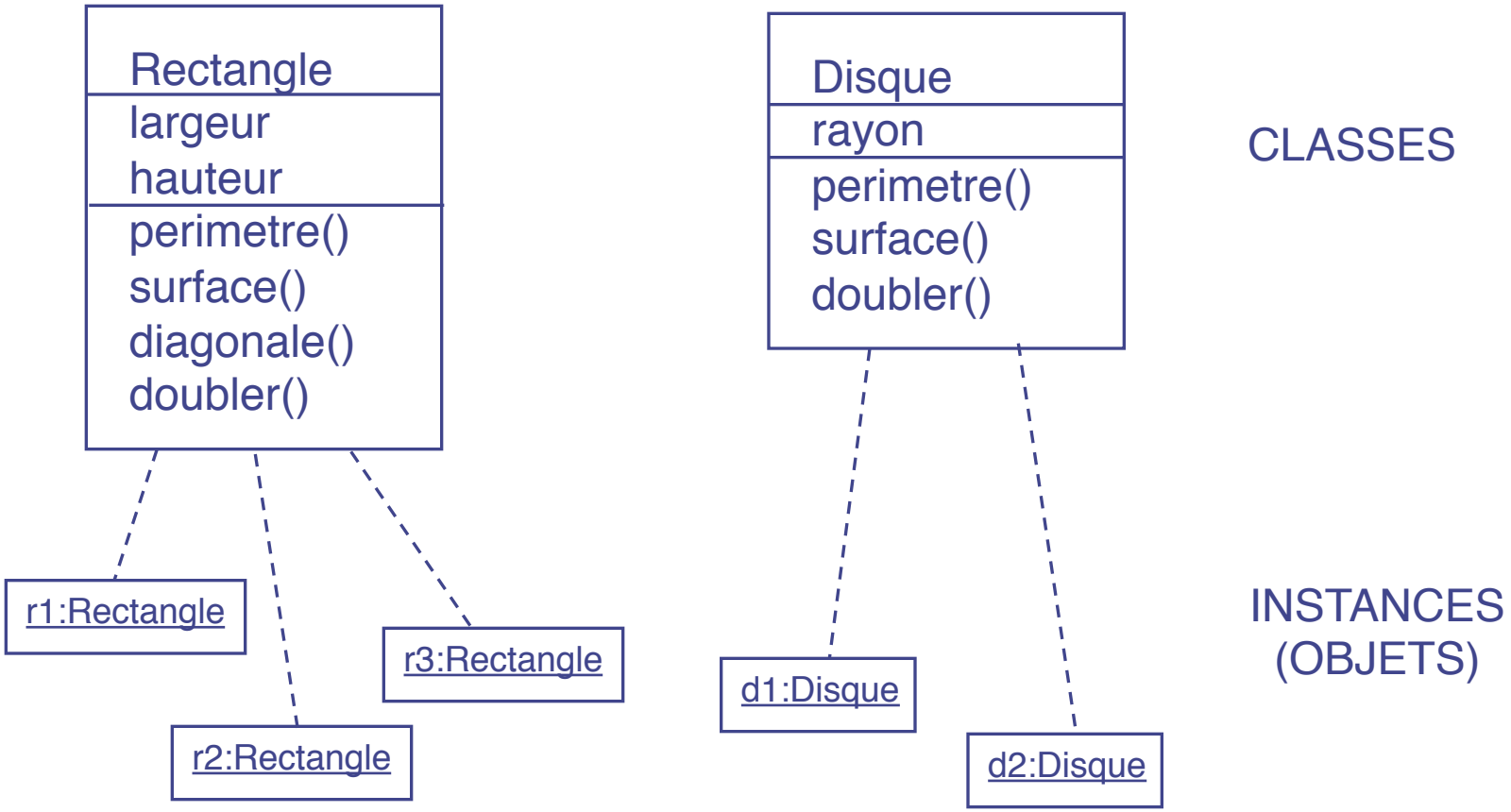
- Les objets du domaine de l'application
  - les objets du monde (réel perçu) du domaine d'application donneront lieu à des objets informatique
- Les artefacts du système informatique:
  - fenêtres, menus, boîtes de dialogue, …
  - événements, transactions, règles

Les classes en Java - rappels

Tout ce que nous avons vu concernant les classes et les objets en UML (voir cours BD) et Python demeure vrai en Java (sauf la syntaxe), à savoir:

- la notion de classe étend celle de type de données
- une classe est un moule pour fabriquer des objets de même structure et de même comportement
- un objet est une instance d'une et d'une seule classe
- une méthode définit une action élémentaire que l'on peut effectuer sur un objet
- un message est une demande d'exécution d'une méthode à un objet

Classes et instances (objets)



Déclaration de classes en Java

- En Java, la déclaration d'une *classe* s'écrit:

```
class Rectangle {
  …
}
```

- Une classe peut posséder des *variables d'instance* (propre à chaque instance):

```
class Rectangle {
double largeur, hauteur;
  …
}
```

- Une classe peut posséder des *variables (ou constantes) de classe*, qui ne seront pas créées dans ses instances. La déclaration est précédée du mot *static*:

```
class Disque {
static final double pi=3.141595;
  …
}
```

Déclaration de classes en Java (suite)

- Les *méthodes* de classe détermine le comportement des instances (objets). La déclaration d'une méthode est composées des éléments suivants:
  - Le type du résultat, ou *void* si la méthode ne produit pas de résultat
  - Le nom de la méthode
  - Le type et le nom des paramètres entre ()
  - Le bloc d'instructions délimité par {}
  - Exemples

```
double perimetre() {return 2*(largeur+hauteur);}
double surface() {return largeur*hauteur;}
double diagonale() {
    return Math.sqrt(largeur*largeur+hauteur*hauteur)}
void doubler() {largeur*=2; hauteur*=2;}
```

Constructeur de classe

- Le *constructeur* de classe est une méthode spéciale qui indique comment initialiser une instance
- Règle de nommage: le constructeur doit porter le même nom que la classe

```
class Rectangle {
        double largeur, hauteur;
        Rectangle(double initL, double initH){        // constructeur d'objets
            largeur=initL;
            hauteur=initH;
        }
        …
}
```

- Ce constructeur initialise les deux variables d'instance *largeur* et *hauteur*
- Constructeur par défaut: si aucun constructeur n'est définit pour une classe, implicitement Java en crée un (mais qui n'initialise pas les variables d'instance)

La classe *Rectangle* complète

```
public class Rectangle {
        double largeur, hauteur;
        public Rectangle(double initL, double initH){        // constructeur d'objets
            largeur=initL;
            hauteur=initH;
        }
        double perimetre() {return 2*(largeur+hauteur);}
        double surface() {return largeur*hauteur;}
        double diagonale() {
            return Math.sqrt(largeur*largeur+hauteur*hauteur)}
        void doubler() {largeur*=2; hauteur*=2;}
}
```

La classe *Disque*

```
class Disque {
    double diametre;
    static final double pi=3.14159;
    Disque(double initD) {diametre=initD;}
    double perimetre() {return pi*diametre;}
    double surface() {return (pi*diametre*diametre)/ 4;}
    double rayon() {return diametre/2;}
    void doubler() {diametre*=2;}
}
```

Les classes dans l'environnement BlueJ

La commande *New Class* de BlueJ génère automatiquement

```
public class Rectangle
{
    private int x; // une variable d 'instance

    public Rectangle() // un constructeur d 'objets Rectangle
    {
        x=0;
    }

    public int sampleMethod(int y) //un exemple de méthode
    {
        return x + y;
    }
}
```

Création d'instances en Java

- Pour créer une instance (objet), on invoque un constructeur avec l'instruction *new.* P.e. :

  new Rectangle(5, 10); // création d'un rectangle de larg. 5, haut. 10

- Pour se référer à l'objet créé, nous avons besoin d'une référence (ou variable). La déclaration d'une référence se fait comme pour les types de base (int, double, char, boolean,…) en faisant précéder le nom de la référence par le nom de la classe:

  Rectangle r1, r2; // déclaration de deux rectangles

- L'assignation se fait au moment de la création de l'instance

  r1 = new Rectangle(5, 10); r2 = new Rectangle(2, 2);

- Il est possible de déclarer la référence et de créer l'objet en même temps:

  Rectangle r1 = new Rectangle(5, 10)

- Dans BlueJ on peut créer des instances « à la main » :
  click droit sur le rectangle de la classe > **new** nomConstructeur(param)

Invocation des méthodes sur une instance

- L'invocation d'une méthode se fait en désignant l'instance (par la référence) et en la faisant suivre du nom de la méthode et des paramètres effectifs données sous forme d'expression, s'il y a lieu:

  instance.méthode(expr1, expr2, …)

- Par exemple, pour obtenir la surface du rectangle r1, on écrira:

  r1.surface()

- Si la méthode possède des paramètres, les expressions passées en paramètre sont évaluées puis affectées aux paramètres correspondants de la méthode

- Si l'on veut invoquer une méthode sur l'instance courante, on écrira:

  this.méthode(expr1, expr2, …)

Un exemple de programme complet

```
class TestRectDisk1{
    public static void main (String args[]) {
      Rectangle r1=new Rectangle(2,4);
      Rectangle r2=new Rectangle(3,4);
      System.out.println("diagonale de r1: "+r1.diagonale());
      System.out.println("périmètre de r2: "+r2.perimetre());
      r2.doubler();
      System.out.println("périmètre de r2: "+r2.perimetre());
      Disque d1,d2; d1=new Disque(2); d2=new Disque(4);
      System.out.println("rayon de d1: "+d1.rayon());
      System.out.println("périmètre de d2: "+d2.perimetre());
      d2.doubler();
      System.out.println("périmètre de d2: "+d2.perimetre()); }
}
```

Déclaration de plusieurs constructeurs

- Il est parfois pratique de proposer plusieurs constructeurs pour une même classe (permettant plusieurs formes de représentations selon les utilisateurs)
- Pour se faire il suffit de définir un nouveau constructeur deavnt se distinguer des autres par le nombre et / ou le type de ses paramètres
- Par exemple, pour la classe Rectangle nous pouvons ajouter un constructeur qui construit un rectangle à partir du coin supérieur gauche (x1,y1) et du coin inférieur droit (x2, y2)

```
class Rectangle {
        double largeur, hauteur;
        Rectangle(double initL, double initH){
            largeur=initL;
            hauteur=initH;
        }
        Rectangle(double x1, double y1, double x2, double y2){
            largeur=Math.abs(x2-x1);
            hauteur=Math.abs(y2-y1);
        }
        …
    }
```

Méthodes de classe

- Nous avons vu qu'il était possible de déclarer des *variables de classe*
- Il est également possible de déclarer des *méthodes de classe*
- Ces méthodes sont destinées à agir sur la classe (plutôt que sur les instances)
- Tout comme pour les variables de classe, la déclaration des méthodes de classe est précédée du mot *static*
- L'invocation à une méthode de classe se fait en faisant précéder le nom de la méthode par celui de la classe

  nom_de_la_classe.méthode(expr1, expr2, …)
- Exemple:

```
class Rectangle {
        static double echelle=1.0;                  // variable de classe
        double largeur, hauteur;
        static void modifierEchelle(double e) {     // méthode de classe
            echelle=e;
        }
    }
```

- Invocation:

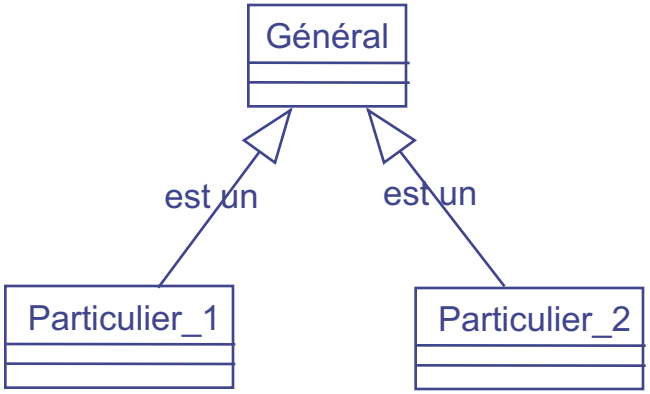
  Rectangle.modifierEchelle(4)

Méthodes de classe: commentaires

- La « célèbre » méthode *main* que nous avons utilisé jusqu'à maintenant pour écrire nos programmes était en fait une *méthode de classe*
- Déclaration:

  static void main(String args[])
- La programmation avec des classes qui contiennent exclusivement des variables de classes et des méthodes de classes nous ramème dans le paradigme de la programmation procédurale

Défintion de sous-classes en Java

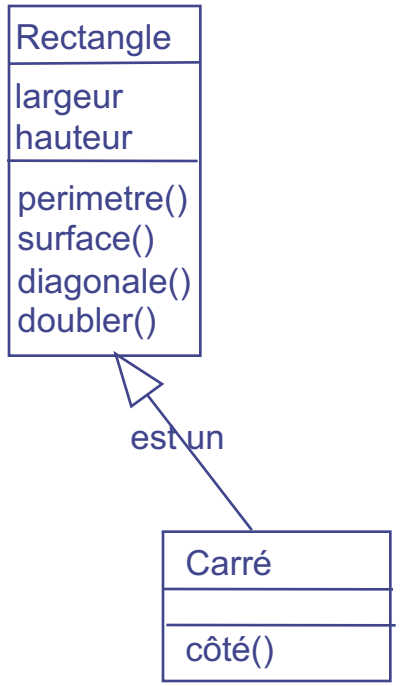


- En Java le mot réservé *extends* permet de déclarer une sous-classe à partir d'une classe de base ou *super-classe*
- La sous-classe *hérite* de tout ce qui a été défini dans la super-classe (variables et méthodes)
- Déclaration:

  class Particulier_1 **extends** Général {… }
  class Particulier_2 **extends** Général {… }
- Le constructeur de la sous-classe peut utiliser le constructeur de la super classe en invoquant *super(…)*

Exemple: *Carre* extension de *Rectangle*

```
class Carré extends Rectangle{
    Carre(double initC){
        super(initC,initC);
    }
    double côté() {return this.hauteur;}
}
```



- *super* est utilisée pour atteindre le constructeur de la super-classe Rectangle

Utilisation de la classe « Carré »

```
class TestCarré {
    public static void main (String args[]) {
        Carré c1=new Carré(4);
        System.out.println("dimension c1: "+c1.largeur+"/"+c1.hauteur);
        System.out.println("surface c1: "+c1.surface());
        System.out.println( "côté de c1 "+c1.côté());
    }
}
```

- Sortie:

```
dimension c1: 4/4
surface c1: 16
coté de c1: 4
```

Assignation et compatibilité des types

- Affectation des objets: supposons qu'une variable (référence) soit de type (classe) C; on peut alors lui affecter un objet de la classe C ou d'une sous-classe de C
- Exemple:

```
Rectangle r;
r = new Carre(4);
```

- Par contre, l'inverse n'est pas possible:

```
Carre c;
c = new Rectangle(5, 0);          // Erreur à la compilation !!!
```

- Lorsqu'on est sûr du type, on peut le changer (garde de type):

```
Rectangle r;
Carre c;
r = new Carre(4);                 // r désigne bien un carré
…
c = (Carre) r;                    // c = r aurait été refusé à la compilation
```

- Attention: à l'exécution un contrôle de type est exécuté et si *r* n'est pas de type carré, un erreur d'exécution se produira

Exercice de déclaration de classe

- Ecrire une classe « tamagotchi » t.q.
  - un tamagotchi est caractérisé par (variables d'instance)
    - un nom (String)
    - un âge (entier)
    - un âge de sagesse (entier)
    - un niveau d'énergie (entier)
  - une variable de classe (c-à-d *static*)
    - *min énergie* valable pour tous les tamagotchis
  - 1 constructeur qui initialise:
    - le nom (passé en paramètre)
    - l'âge (à 0)
    - l'âge de sagesse (de 20 à 30, passé en paramètre)
    - le niveau d'énergie (à 8)
  - une méthode manger (qui augmente l'énergie)
  - une méthode vieillir (qui augmente l'âge et diminue l'énergie)

  - *voir l'énoncé complet : série 3, exercice 4*

Rappel de la référence

- En Java:
  - 8 types primitifs: boolean, char, int, float, double…
  - tous les autres types sont des types « référence », comprenant les classes (comme les chaînes de caractères) et les tableaux (array)
  - le but d'une variable référence et de référencer un objet, c'est-à-dire un moyen de désigner un objet
- une variable de type référence stocke l 'adresse mémoire où un objet réside
- on appelle plus simplement « référence » une variable de type référence
- si une référence ne se réfère à aucun objet, sa valeur est « **null** »

La référence : exemple

- Déclaration de la classe Tamagotchi (voir série 3 d'exercices Java)

```
public class Tamagotchi {
String nom; // variable d 'instance
Int age;       //          ‘’
…
public Tamagotchi(String n) // constructeur de tamagotchi
…
}
```

- déclaration de trois références et instanciations de deux objets:

```
Tamagotchi t1 = new Tamagotchi(‘’Louis‘’);
Tamagotchi t2 = new Tamagotchi(‘’Louis‘’);
Tamagotchi t3 = t1;
```

La référence : exemple (suite)

- Après ces instructions, il y a en mémoire:
  - 3 références (t1, t2 et t3)
  - 2 objets de type tamagotchi (avec le même nom: *Louis*)

  - t1 **et** t3 se réfèrent tous les deux au premier objet tamagotchi
  - t2 se réfère au deuxième objet tamagotchi

Opérateurs pour les références

- En Java, il existe seulement **3** opérateurs pour les références:
  - **=** pour l 'affectation
  - **==** et **!=** pour tester l 'égalité resp. l 'inégalité
- Toutes les autres opérations souhaitées sont à définir avec des méthodes

- Exception: les références de type String possède deux opérateurs supplémentaires: + et += (concaténation de chaînes de caractères)

Référence: la signification de =

- La signification de = (affectation) pour les références est exactement la même que pour les types primitifs: la **valeur contenue** dans la référence de droite est recopiée dans la référence de gauche
- comme la référence contient l ’**adresse mémoire** où réside l ’objet référencé, c ’est cette **adresse** qui est recopiée dans la référence affectée (et non les valeurs stockées dans l ’objet !)

Référence: la signification de == et !=

- L ’expression **t1==t2**
- teste si la valeur (référence) contenue dans la variable t1 est égale à la valeur contenue dans t2
- Comme les valeurs sont des références adresses mémoire d ’objets, cela revient à tester si d1 et d2 se réfèrent au **même** objet
- L’opérateur **!=** teste l ’inégalité

Référence: la signification de == et != (suite)

- Pour tester l ’égalité du contenu de deux objets, on utilise la méthode

  equals

  définie sur dans la classe Object

Exemple

- Soit les déclarations:

```
Tamagotchi t1 = new Tamagotchi(‘’Louis‘’);
Tamagotchi t2 = new Tamagotchi(‘’Louis‘’);
Tamagotchi t3 = t1;
```

- que valent les expressions suivantes ?

```
t1 == t2                // rép: …
t1 != t2                // rép: …
t1 == t3                // rép: …
t1.equals(t2)           // rép: …
```

Exemple

- Soit les déclarations:

```
Tamagotchi t1 = new Tamagotchi(‘’Louis‘’);
Tamagotchi t2 = new Tamagotchi(‘’Louis‘’);
Tamagotchi t3 = t1;
```

- que valent les expressions suivantes ? Réponses:

```
t1 == t2              // faux car t1 et t2 référencent des objets diff.
t1 != t2              // vrai
t1 == t3              // vrai car t1 et t3 référencent le même objet
t1.equals(t2)         // vrai car l’objet référencé par t1 et l ’objet
                      // référencé par t2 contiennent la même valeur
                      // (dans la variable nom)
```