Chapitre II
Classes et objets (rappels)

(mais pas d'héritage)

ce dont on ne parlera pas…

- Syntaxe
  - expressions:
    - variables, opérateurs, invocation de méthodes
    - une expression a un type et une valeur
  - instructions:
    - `expression;`
    - déclarations
    - contrôle de flot
      - if, if else
      - switch
      - while, do while, for
      - break, continue, return
    - blocs:
      - { instructions …}

ce dont on ne parlera pas…

- catégories de variables:
  - variables d'instances
  - variables de classes (static)
  - variables locales
  - paramètres de méthodes
    - toute variable a un type qui est défini par sa déclaration (portée)

Classes et objets

- I) Introduction
- II) Classe: membres et modificateurs
- III) Champs: modificateurs
- IV) Vie et mort des objets, Constructeurs
- V) Méthodes
- VI) Exemple

I) Introduction

- Classe
  - Regrouper des données et des méthodes (encapsulation)
    - Variables de classe
    - Méthodes de classe
  - Classes<->type
- Objet (ou instance)
  - Résultat de la création d'un objet
    - Variables d'instance
    - Méthodes d'instance
- Toute classe hérite de la classe `Object`

Variable et référence

- type référence - type primitif

```
int i=0;
```

  - i est une variable d'un type primitif: i contient une

Références

- Une variable est (pour un type référence):
  - une référence à un objet (créé par new)
    - `null` : référence universelle
  - `this` est une auto-référence:

```
public class Point {
    public int x = 0;
    public int y = 0;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Références

- La valeur d'une variable est (pour un type référence) une référence:
  - conséquences:
    - dans le passage par valeur un type référence correspond à un passage par référence
    - 'a == b' teste si les a et b référencent le même objet
    - Méthode `equals` qui peut être redéfinie (défaut this==obj)

```
int i=0;
int j=0;
(i==j) // vrai
class A{
    int i=0;
}
A a;
A b=new A();
a=b;
(a==b) // vrai
b=new A();
(a==b) // faux
```

primitif et référence

- type « wrapper »:
  - `int` est un type primitif `Integer` est un type référence qui lui correspond
  - plus généralement à chaque type primitif correspond un type wrapper
    - `boolean` `Boolean`
    - `byte` `Byte`
    - `char` `Character`
    - `float` `Float`
    - `int` `Integer`
    - `long` `Long`
    - `short` `Short`
    - `double` `Double`

primitif et référence

- conversions automatiques:
  - primitif vers référence: autoboxing (affectation et passage de valeur)
  - référence vers primitif: unboxing (affectation et passage de valeur)

```
public static int f(Integer i){return i;}
public static Integer g(int i){return i;}
public static void main(String[] args) {
    int i= new Integer(5);
    Integer n=6;
    System.out.println(f(3));
    System.out.println(g(n));
}
```

II) Classes

déclaration:

```
public class A {…}
```

modificateur nom corps

(plus `extends implements`)

membres d 'une classe sont:

- Champs = données
- Méthodes = fonctions
- déclaration de classe (internes)

Modificateur de classe

- Précède la déclaration de la classe
  - Annotations (plus tard…)
  - `public` (par défaut package)
    - une seule class public par fichier de même nom que le fichier .java
  - `abstract` (incomplète, pas d'instance)
  - `final` (pas d'extension)
  - (`Strictfp` (pour les calculs en float))

III) Champs

- Modificateurs
  - annotations
  - modificateur d'accès
    - `private` (uniquement dans la classe)
    - `protected` (dans la classe et les sous-classes)
    - `public` (accessible à tous)
    - `package` (défaut: accessible à toutes les classes du package)
  - `static` (variables de classe)
  - `final` (constantes)
  - `transient`
  - `volatile`
- Initialisations
- Création par opérateur `new`

IV) Vie et mort des objets, constructeurs

- Création d'une instance: opérateur new
- Objet mort = plus aucune référence à cet objet -> garbage collector
  - on peut exécuter du code spécifique quand un objet est détruit :
    - méthode
      `protected void finalize() throws Throwable`

Constructeurs

- Appelés par l'opérateur new pour créer un objet
  - Peuvent avoir des paramètres (avec surcharge)
  - Initialisent les objets
  - Constructeur par défaut (si aucun constructeur n'est défini)
  - (this() dans un constructeur appelle le constructeur

```
public class Rectangle {
    private int x, y;
    private int width, height;
    public Rectangle(int x, int y, int width, int height) {
        this.x=x; this.y=y; this.width=width; this.height=height;
    }
    public Rectangle() {
        this(0, 0, 1, 1);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
```

  -)

Exemple:

```
public class Astre {
    private long idNum;
    private String nom = "<pasdenom>";
    private Astre orbite = null;
    private static long nextId = 0;
    /** Creation d'une nouvelle instance of Astre */
    private Astre() {
        idNum = nextId ++;
    }
    public Astre(String nom, Astre enOrbite){
        this();
        this.nom=nom;
        orbite=enOrbite;
    }
    public Astre(String nom){
        this(nom,null);
    }//…
```

Statique - dynamique (bis)

- Statique <-> à la compilation
- Dynamique <-> à l'exécution
- Le type d'une variable est déterminé à la compilation (déclaration et portée)
- Avec la possibilité de l'héritage une variable peut être une référence sur un objet d'un autre type que le type de sa déclaration

Static ou non

- Une variable (une méthode) déclarée static est une variable (méthode) de classe: elle est associée à la classe (pas à une instance particulière).
- Statique parce qu'elle est créée au moment de la compilation (pas de new()).
- Statique -> les initialisations doivent avoir lieu à la compilation.
- une variable d'instance est propre à chaque instance
- une méthode d'instance a accès à l'état de l'instance

Initialisations

- initialisation d'une variable statique:

```
private static long nextId = 0;
```

- Bloc d'initialisation:

```
public class Puissancedeux {
    static int[] tab = new int[12];
    static{
        tab[0]=1;
        for(int i=0; i< tab.length-1;i++)
            tab[i+1]= suivant(tab[i]);
         }
    static int suivant(int i){
        return i*2;
    }
}
```

- (l'initialisation d'une variable d'instance n'est pas une « vraie » initialisation, pourquoi?)

V) Méthodes

- Modificateurs:
  - Annotations
  - Contrôle d'accès (comme pour les variables)
  - `abstract` (le code n'est pas défini)
  - `static` : méthode de classe -> n'a pas accès aux variables d'instances
  - un constructeur est une méthode d'instance particulière
  - `final` ne peut pas être remplacée
  - `synchronized`
  - `native` (utilisation de fonctions « natives »)
  - `strictfp`

Surcharge

- Un même nom de fonction pour plusieurs fonctions qui sont distinguées par leur signature

(Java, C++, Ada permettent la surcharge)

En C '/' est surchargé

3/2 division entière -> 1

3.0/2 division réelle -> 1,5

Surcharge

- la signature d'une méthode est son nom et ses paramètres

  ```
  int f(int x, String s, A a)
  ```

  signature: *f (int, String , A)*

- un même nom de méthode peut correspondre à plusieurs codes s'ils ont des signatures différentes

```
...
                    (          )
                    (       )
                    (          )
                    (       ,          )
```

Surcharge

```
 public int f(int i){
        return i;
    }
//    public double f(int i){
//        return Math.sqrt( i);
//    }
    public int f(double i){
        return (int) Math.sqrt( i);
    }
    public int f(char c){
        return c;
    }
```

Passage par valeur

```
public class ParamParVal {
    public static void parVal(int i){
        i=0;
        System.out.println("dans parVal i="+i);
    }
}
//…
 int i =100;
 System.out.println("Avant i="+i);
 ParamParVal.parVal(i);
 System.out.println("Avant i="+i);
---------------
Avant i=100
dans parVal i=0
Avant i=100
```

dans la passage par valeur, la valeur du paramètre est copiée.

Mais…

- Comme les variables sont de références (sauf les types primitifs)…

```
public static void bidon(Astre a){
    a=new Astre("bidon", null);
    System.out.println("bidon a="+a);
}
public static void bidonbis(Astre a){
     a.setNom("bidon");
     a.setOrbite(null);
    System.out.println("bidonbis a="+a);
}
```

Méthodes…

- Contrôler l'accès:
  - a priori pas d'accès direct à une variable d'instance-> contrôle par de méthodes d'accès

```
//…
    public void setNom(String n){
        nom=n;
    }
    public void setOrbite(Astre a){
        orbite=a;
    }
    public String getNom(){
        return nom;
    }
    public Astre getOrbite(){
        return orbite;
    }
```

Méthodes, remplacement…

```
public String toString(){
    String st=idNum + "("+nom+")";
    if (orbite != null)
        st += "en orbite "+ orbite;
    return st;
}
```

Remplace la méthode toString de la classe Object.

Toute classe hérite de Object

Nombre variable d'arguments…

```
public static void affiche(String ... list){
        for(int i=0;i<list.length;i++)
            System.out.print(list[i]+" ");
}

//…

affiche("un", "deux","trois");
```

Méthodes main

```
 public static void main(String[] args) {
        for(int j =0; j<args.length;j++){
            System.out.print(args[j] + " ");
        }
}
```

Le main est le point d'accès et peut avoir des arguments:

```
public class Astre {
    private long idNum;
    private String nom = "<pasdenom>";
    private Astre orbite = null;
    private static long nextId = 0;
    private Astre() { idNum = nextId ++;}
    public Astre(String nom, Astre enOrbite){
        this();
        this.nom=nom;
        orbite=enOrbite;
    }
    public Astre(String nom){this(nom,null);}
    public Astre(Astre a){
        idNum = a.idNum;
        nom=a.nom;
        orbite=a.orbite;
    }
    public void setNom(String n){nom=n;}
    public void setOrbite(Astre a){orbite=a;}
    public String getNom(){ return nom;}
    public Astre getOrbite(){return orbite;}
    public String toString(){
        String st=idNum + "("+nom+")";
        if (orbite != null)
            st += "en orbite "+ orbite;
        return st;
    }
}
```