Cours n°3

rappels

Entrée-sortie

```
public static void main(String[] args) {
     // sortie avec printf ou
     double a = 5.6d ;
     double b = 2d ;
     String mul = "multiplié par" ;
     String eq="égal";
     System.out.printf(Locale.ENGLISH,
                "%3.2f X %3.2f = %6.4f \n", a ,b , a*b);
     System.out.printf(Locale.FRENCH,
           "%3.2f %s %3.2f %s %6.4f \n", a, mul,b eq,a*b);
      System.out.format(
         "Aujourd'hui %1$tA, %1$te %1$tB,"+
          " il est: %1$tH h %1$tM min %1$tS \n",
                Calendar.getInstance());
       // System.out.flush();
```

Sortie

5.60 X 2.00 = 11.2000

5,60 multiplié par 2,00 égal 11,2000

Aujourd'hui mardi, 10 octobre, il est: 15 h 31 min 01

Scanner

```
Scanner sc = new Scanner(System.in);
for(boolean fait=false; fait==false;){
    try {
       System.out.println("Répondre o ou O:");
       String s1 =sc.next(Pattern.compile("[Oo]"));
              fait=true;
    } catch(InputMismatchException e) {
    sc.next();
    }
}
if (sc.hasNextInt()){
    int i= sc.nextInt();
    System.out.println("entier lu "+i);
}
System.out.println("next token :"+sc.next());
sc.close();
```

Scanner

```
if (sc.hasNextInt()){
    int i= sc.nextInt();
    System.out.println("entier lu "+i);
}
System.out.println("next token :"+sc.next()); sc.close();
String input = "1 stop 2 stop éléphant gris stop rien";
Scanner s = new(Scanner(input).useDelimiter("\\s*stop\\s*");
        System.out.println(s.nextInt());
        System.out.println(s.nextInt());
        System.out.println(s.next());
        System.out.println(s.next());
        s.close();
    }
```

Sortie

- next token :o
- 1
- 2
- éléphant gris
- rien

Les classes…

- System
  - System.out variable (static) de classe PrintStream
    - PrintStream contient print (et printf)
  - System.in variable (static) de classe InputStream
- Scanner

Chapitre II
Classes et objets (rappels)

(mais pas d'héritage)

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
    - Variables de classe
    - Méthodes de classe
  - Classes<->type
- Objet (ou instance)
  - Résultat de la création d'un objet
    - Variables d'instance
    - Variables de classe
- Toute classe hérite de la classe `Object`

II) Classes

- Membres d 'une classe sont:
  - Champs = données
  - Méthodes = fonctions
  - Classes imbriquées

Modificateur de classe

- Précède la déclaration de la classe
  - Annotations (plus tard…)
  - `public` (par défaut package)
  - `abstract`(incomplète, pas d'instance)
  - `final`(pas d'extension)
  - `Strictfp` (technique…)

III) Champs

- Modificateurs
  - annotations
  - Contrôle d'accès
    - `private`
    - `protected`
    - `public`
    - `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 de référence à cet objet -> garbage collector
  - on peut exécuter du code spécifique quand un objet est détruit :
    `protected void finalize() throws Throwable`

Références

- Une variable est (en général) une référence à un objet
  - Type primitif: directement une valeur
  - Type référence : une référence à un objet (existant ou créé par new)
    - null : référence universelle
    - 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)

Exemple

```
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
```

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)
  - Constructeur de copie

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);
    }//…
```

Exemples…

- Copie

```
public Astre(Astre a){
        idNum = a.idNum;
        nom=a.nom;
        orbite=a.orbite;
    }
```

Statique - dynamique

- 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

- 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 peut être créée au moment de la compilation (pas de new()).
- Statique -> les initialisations doivent avoir lieu à la compilation.

Initialisations

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

- Bloc d'initialisation

```
private static long netxId = 0;
{
        idNum = nextId++;
 }
```

Initialisation static

```
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;
    }
}
```

V) Méthodes

- Modificateurs:
  - Annotations
  - Contrôle d'accès (comme pour les variables)
  - `abstract`
  - `static` n'a pas accès aux variables d'instances
  - `final` ne peut pas être remplacée
  - `synchronized`
  - `native` (utilisation de fonctions « native »)
  - `strictfp`

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
```

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:

```
//…
    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

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:

VI) exemple: Les astres…

```
package exempleclasses;

/**
 *
 * @author sans
 */
public class Astre {
    private long idNum;
    private String nom = "<pasdenom>";
    private Astre orbite = null;
    private static long nextId = 0;
    /** Creates a new instance of Astre */
    private Astre() {
        idNum = nextId ++;
    }
```

Suite

```
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;
    }
}
```

Chapitre III
Héritage

Chapitre III: Héritage

- A) Extensions généralités
  - Affectation et transtypage
- B) Méthodes
  - Surcharge et signature
- C) Méthodes (suite)
  - Redéfinition et liaison dynamique
- D) Conséquences
  - Les variables
- E) Divers
  - Super, accès, final
- F) Constructeurs et héritage

A) Extension: généralités

- Principe de la programmation objet:
  - un berger allemand est un chien
    - il a donc toutes les caractéristiques des chiens
    - il peut avoir des propriétés supplémentaires
    - un chien est lui-même un mammifère qui est lui-même un animal: hiérarchie des classes
  - On en déduit:
    - Hiérarchie des classes (Object à la racine)
    - et si B est une extension de A alors un objet de B est un objet de A avec des propriétés supplémentaires

Extension: *généralités*

Quand B est une extension de la classe A:

- Tout objet de B a toutes les propriétés d'un objet de A (+ d'autres).
- Donc un objet B peut être considéré comme un objet A.
- Donc les variables définies pour un objet de A sont aussi présentes pour un objet de B (+ d'autres). (Mais elles peuvent être *occultées*)
- Idem pour les méthodes : Les méthodes de A sont présentes pour B et un objet B peut définir de nouvelles méthodes.
- Mais B peut *redéfinir* des méthodes de A.

Extension de classe

Si B est une extension de A

- pour les variables:
  - B peut ajouter des variables (et si le nom est identique cela *occultera* la variable de même nom dans A)

    (occulter = continuer à exister mais "caché")
  - *Les variables de A sont toutes présentes pour un objet B*, mais certaines peuvent être cachées
- pour les méthodes
  - B peut ajouter de nouvelles méthodes
  - B peut redéfinir des méthodes (même signature)

Remarques:

- pour les variables
  - c'est le nom de la variable qui est pris en compte (pas le type ni les droits accès).
  - dans un contexte donné, à chaque nom de variable ne correspond qu'une seule déclaration.
  - (l'association entre le nom de la variable et sa déclaration est faite à la compilation)
- pour les méthodes
  - c'est la signature (nom + type des paramètres) qui est prise en compte:
    - on peut avoir des méthodes de même nom et de signatures différentes (surcharge)
    - dans un contexte donné, à un nom de méthode et à une signature correspond une seule définition
    - (l'association entre le nom de la méthode et sa déclaration est faite à la compilation, mais l'association entre le nom de la méthode et sa définition sera faite à l'exécution)

Extension (plus précisément)

- Si B est une extension de A (`class B extends A`)
  - Les variables et méthodes de A sont des méthodes de B (mais elles peuvent ne pas être accessibles: `private`)
  - B peut ajouter de nouvelles variables (si le *nom* est identique il y a occultation)
  - B peut ajouter des nouvelles méthodes si la *signature* est différente
  - B redéfinit des méthodes de A si la signature est identique