La programmation objet
- implémentation avec Java/C# -

Bachir Djafri
Lab. IBISC / Dép. Informatique
Université d'Évry Val d'Essonne
bachir.djafri@ibisc.univ-evry.fr
http://www.ibisc.univ-evry.fr/~djafri

Plan

- Rappels
  - Programmation de base avec Java et C#
  - Structure des langages objet (Java/C#)
  - Les caractéristiques et les outils
- L'approche objet (en théorie)
- La programmation objet avec Java et C#
  - Objets et Classes
  - Envoi de message
  - Héritage
- *Références*

Langages et outils programmation Objet

Bachir Djafri
Lab. IBISC / Dép. Informatique
Université d'Évry Val d'Essonne
bachir.djafri@ibisc.univ-evry.fr
http://www.ibisc.univ-evry.fr/~djafri

Les commentaires

- Les caractères Unicode
  - 16 bits, `'\u0000'` à `'\uFFFF'`
- Trois types de commentaires
  - `// un commentaire sur une ligne`
  - `/* un autre commentaire sur deux lignes */`
  - `/// un commentaire de documentation pour C#`
  - `/** un commentaire de documentation pour Java */`

Les variables

- Variable = nom + type
- Déclaration de variables :
  - type nom ;
  - Exemples : `int var ;  char c ;` …
- 2 classes de types :
  - Primitifs : une seule valeur simple
  - Références : types composés
- Porté et visibilité des variables (identificateurs) : bloc

Les types de données primitifs (1)

- Type **`boolean`** (1 bit), **`true`** ou **`false`**
- Type **`char`** (2 octets), caractères Unicode, de `'\u0000'` à `'\uFFFF'`
- Les entiers signés :
  - **`byte`** (1 octet), -128 à +127
  - **`short`** (2 octets), -32768 à +32767
  - **`int`** (4 octets),
  - **`long`** (8 octets),
- Les réels à virgule flottante :
  - **`float`** (4 octets), IEEE 754-1985
  - **`double`** (8 octets). IEEE 754-1985

Les types de données primitifs (2)

- Les variables peuvent être déclarées n'importe où dans un bloc.
  Exemples Java:

```
int i = 0xFF; // notation hexadécimale,                    i = 255
int j = 0377; // notation octale,                          j = 255
long k = 1234L; // valeur de type long, 12341             k = 1234
long l = 0XffL; // valeur de type long en notation hexa.   l = 255
float f = 1234.56F; // valeur de type float               f = 1234.56
double d = 123.45e3; // valeur de type double            d = 123450.0

final byte b1 = 127 , b2 = 1;
int  sum = b1 + b2; // OK, sum = 128                      sum = 128
byte sum = (byte)(b1 + b2); // OK, avec sum = -128        sum = -128
```

- Une variable déclarée **`final`** (**`const`** en C#) ne peut pas changer de valeur après son initialisation : constante.

Les opérateurs

- Les opérateurs arithmétiques : `+, -, *, /, %`
- Les opérateurs d'incrémentation/décrémentation : `++, --`
- Les opérateurs relationnels : `<, >, <=, >=, ==, !=`
- Les opérateurs logiques : `&&, ||, ! , &, |, ^`
- Les opérateurs de bits : `<<, >>, >>>, &, |, ^, ~`
- Les opérateurs d'affectation : `=, +=, -=, *=, %=, …`
- L'opérateur conditionnel : `?:`

Expression et instruction

- Expression : variables + opérateurs = valeur
  - Pas d'action (en général)
- Instruction : réalise une action
  - Pas de valeur (en général)
- Java
  - Expression/instruction
    - Exemple : `var++; a=b=c=3*d; i=5*v--;`
- Expressions mixtes : opérandes de types différents
    - Exemple : `i*f*d; (int i; float f; double d;)` : type = ?

Conversions de type

- Les affectations non implicites doivent être *castées* (sinon erreur à la compilation).
  - Opérateur de *cast* : `(type)` un par type
- Les cas de conversion permis (implicites) :
  - **byte ⟹ short ⟹ int ⟹ long ⟹ float ⟹ double**
- Exemples (conversion forcée par une affectation) :

```
long l = i;   // ok, i est une variable entière
byte b = i;   // error: Explicit cast needed to convert int to byte
byte b = 258; // error: Explicit cast needed to convert int to byte
byte b = (byte)i; // ok, utilisation de l'opérateur de cast (perte d'info)
```


Les instructions de contrôle

- Possibilité d'effectuer des choix : selon des conditions
  - Calcul ⟹ selon le résultat ⟹ traitement différent
- Possibilité de faire des boucles (répétitions, itérations)
  - Tant que le traitement n'est pas fini
    faire …
- Possibilité de faire des branchements (*goto*)
  - Étiquettes et instructions étiquetées
- Possibilité de traiter les erreurs : exceptions

Les instructions de contrôle

- Essentiellement les mêmes qu'en C et les autres langages
  - **`if-else`**, **`switch-case`**, **`while`**, **`do-while`**, **`for`**
  - instruction sous forme d'expression : `=`, `++`, `+=`, `*=`, `...`
  - instruction vide (;) & instruction composée (*bloc*)
- Les instructions étiquetées
  - Les étiquettes sont utilisées par les instructions **break** et **continue**

```
UN: while(cond1) {
  DEUX: for(exp1;exp2;exp3) {
      TROIS: while(cond2) {
              if (cond3) continue UN;
              if (cond4) break DEUX;
              continue;
      }
  }
}
```

Les instructions de contrôle

- L'instruction conditionnelle `if-else`
- Syntaxe :

```
if(condition) Instruction
if(condition) Instruction  else  Instruction
```

- Exemples :

```
if(i>0) y=x/i; else { x=i; y +=x; }

if(a>b) if(b>c) a-=c ; else a-=b;
   //Un else se rapport toujours au dernier if rencontré //
   auquel un else n'a pas été attribué.
```


Les instructions de contrôle

- L'instruction sélective `switch-case`
- Syntaxe :

```
switch(exp){
  case exp_cst1 : Instructions1
  case exp_cst2 : Instructions2
  ...
  [default : Instruction(s)]
}
```

- Exemple :

```
switch (mois) {
  case 12 : nbrJours += 31;
  case 11 : nbrJours += 30;
  ... }
```


Les instructions de contrôle

- L'instruction itérative `while`
- Syntaxe :

```
while (condition) Instruction
while (condition) { Instruction(s) } //instruction bloc
```

- Exemples :

```
int i=0 ; int somme=0 ;
while (i++ < 10) somme+=i; /*somme = 55 */

while (i<10){ somme=somme+i; i=i+1; } /*somme = 45 */
```


Les instructions de contrôle

- L'instruction itérative `do-while`
- Syntaxe :

```
do{ instruction(s) } while(condition);
```

- Exemples :

```
int i=0 ; int somme=0 ;
do{ somme+=i; } while (i++ < 10); /*somme=55*/

do{ somme=somme+i; i=i+1; } while (i<10);
/*somme = 45 */
```

Les instructions de contrôle

- L'instruction itérative `for`
- Syntaxe :

```
for(exp1 ; exp2 ; exp3) Instruction
for(exp1 ; exp2 ; exp3){ Instruction(s) } //bloc
```

- Exemples :

```
int i=0 ; int somme=0 ;
for(i=0 ; i<10 ; i++) somme+=i;

for(i=0 ; i<10 ; ){ somme+=i; i++; }
```


Les instructions de contrôle

- Les instructions de rupture de séquences
- Syntaxe :

```
break [label]
continue [label]
label: Instruction
return [exp];
```

- Exemple :

```
switch (mois) {
  case 1 : nbrJours = 31; break;
  case 2 : nbrJours = 28; break;
  ... }
```


Les instructions de contrôle

- Les instructions étiquetées
  - Les étiquettes sont utilisées par les instructions **break** et **continue**

```
UN: while(cond1) {
   DEUX: for(exp1;exp2;exp3) {
     TROIS: while(cond2) {
          if (cond3) continue UN; //Reprend sur la première boucle while
          if (cond4) break DEUX; // Sort de la boucle for si cond4 vraie
          continue;              // Reprend sur la deuxième boucle while
     }
   }
}
```


Les méthodes

- Méthode = collection nommée d'instructions
- Équivalente à une fonction C
- Signature d'une méthode
  - Modificateurs
  - Type de la valeur retournée par la méthode
  - Nom de la méthode
  - Type et nom des paramètres de la méthode
  - *Type des exceptions que la méthode peut soulever*
- Corps de la méthode = instruction composée (suite d'inst.)
- Invocation d'une méthode = expression/instruction

Exemples de méthodes

```
public class MonPremierProgramme {

    // méthode principale, la seule 'exécutée' par
    // la machine virtuelle Java
    public static void main(String[] args) {
        int i = 3;
        double d = 7.3;
        System.out.println(d + " puissance " + i +" = " +
                puissance(d, i) );
    }

    // une autre méthode appelée par la méthode principale
    static double puissance(double d, int i) {
        if(i<=0) return 1;
        else return d*puissance(d, i-1);
    }
}
```


Les exceptions

- Permettent de séparer un bloc d'instructions de la gestion des erreurs pouvant survenir dans ce bloc.

```
try {
    // Code java pouvant lever des Exceptions (IOException,
    // SecurityException, ...)
} catch (IOException e) {
    // Gestion des IOException et des sous-classes de IOException
} catch (Exception e){
    // Gestion des autres exceptions
} finally {
    // code optionnel exécuté dans tous les cas
}
```


Les entrées/sorties

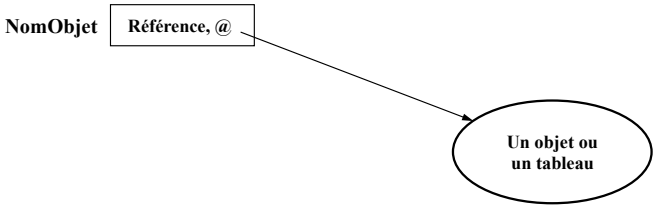
```
public class MonPremierProgrammeJava {

   public static void main(String[] args) {

      int i = 5;

      /* instruction de sortie */

      System.out.println(" Une chaîne de caractères ");

      System.out.print(" la variable i = " + i );
   }

}
```


Les entrées/sorties

```
public class MonPremierProgrammeCsharp {

   public static void Main() {

      int i = 5;

      /* instruction de sortie */

      System.Console.WriteLine("Une chaîne.. ");

      System.Console.Write("La variable i = " + i);
      System.Console.Write("La variable i = {0} ", i);
   }

}
```

Les types références

- Les tableaux, les classes (String) et les interfaces sont des références : adresses, pointeurs (# pointeurs C)



Les tableaux

- Déclaration

```
int[] tableauEntiers; // équivalent à int tableauEntiers[]; en langage C
Color cubeRGB[][][];  // une autre notation possible
```

- Création et initialisation

```
tableauEntiers = new int[35];
cubeRGB = new Color[64][64][64];
int[] T = {1, i, 3, 5*j, 7, 5+11};
tableauEntiers[13] = 51;
```

- Manipulation

```
int l = tableauEntiers.length; // l = 35
int e = tableauEntiers[45]; // java.lang.ArrayIndexOutOfBoundsException
```


Les chaînes de caractères

- Déclaration

```
String message; // équivalent à : char *message; en langage C
```

- Création et initialisation

```
String message = new String("Bonjour");
String message2 = message; // message2 pointe sur le même objet que message
String vide = ""; // chaîne vide
String msg = "Hello!";
String s = msg + message; // s = "Hello! Bonjour"
```

- Manipulation (voir classe String)

```
int l = msg.length(); // l = 6
char c = msg.charAt(4); // c = 'o';
```


L'approche objet

Bachir Djafri
Lab. IBISC / Dép. Informatique
Université d'Évry Val d'Essonne
bachir.djafri@ibisc.univ-evry.fr
http://www.ibisc.univ-evry.fr/~djafri

Concepts fondamentaux de l'approche objet

- Objet = État + Comportement + Identité
- Classe (type)
- Communication entre objets par envoi de messages
- Héritage (Liens de parenté)
- Polymorphisme

Les objets

- Les objets (du monde réel) nous entourent ; ils naissent, vivent et meurent
- Les objets informatiques définissent une représentation simplifiée des entités du monde réel
- Les objets représentent des entités
  - concrètes : avec une masse
  - abstraites : concept

Objet et abstraction

- Une abstraction est un résumé, un condensé
- Mise en avant des caractéristiques essentielles
- Dissimulation des détails (complexité)

- Une abstraction se définit par rapport à un point de vue
  - Exemples d'abstractions
    - Une carte routière
    - Un nombre complexe
    - Un téléviseur
    - Une transaction bancaire
    - Une porte logique
    - Une pile
    - Un étudiant

L'identité

- Tout objet possède une identité qui lui est propre et qui le caractérise
- L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état
- Les langages objets utilisent généralement les adresses (*références, pointeurs)* pour réaliser les identifiants

  Rq : un attribut ***identifiant*** *n'est pas nécessaire*

L'état

- L'état d'un objet :
  - regroupe les valeurs instantanées de tous les attributs d'un objet
  - évolue au cours du temps
  - est la conséquence des comportements passés (à un instant)
- Exemples
  - un signal électrique : l'amplitude, la pulsation, la phase, …
  - une voiture : la marque, la puissance, la couleur, le nombre de places assises, …
  - un étudiant : le nom, le prénom, la date de naissance, l'adresse, …

Le comportement

- Le comportement d'un objet
  - décrit les actions et les réactions d'un objet
  - regroupe toutes les *compétences* d'un objet
  - se représente sous la forme d'opérations (méthodes)
- Un objet peut faire appel aux compétences d'un autre objet
- L'état et le comportement sont liés
  - Le comportement dépend de l'état (en général)
  - L'état est modifié par le comportement (en général)

Communication entre objets

- Application = collection d'objets collaborant
- Les objets travaillent en synergie afin de réaliser les fonctions de l'application
- Le comportement global d'une application repose sur la communication entre les objets qui la composent
- Les objets
  - ne vivent pas en ermites
  - Les objets interagissent les uns avec les autres
  - Les objets communiquent en échangeant/envoyant des messages

Communication (suite)

- Catégories de messages (méthodes) :
  - **Constructeurs** : créent des objets
  - **Accesseurs** : renvoient tout ou partie de l'état
  - **Modifieurs** : changent tout ou partie de l'état
  - ***Destructeurs*** *:* *détruisent des objets*
  - **Itérateurs** : parcourent une collection d'objets

Les classes

- La classe
  - est une description abstraite d'un ensemble d'objets
  - peut être vue comme la factorisation des éléments communs à un ensemble d'objets
  - décrit le domaine de définition d'un ensemble d'objets
- Description des classes
  - Séparée en deux parties
    - La **spécification** d'une classe : décrit le domaine de définition et les propriétés des instances de cette classe (type de donnée)
    - La **réalisation** : décrit comment la spécification est réalisée

Conclusion

- Les objets naissent, vivent et meurent
- Les objets interagissent entre eux par envoi de messages
- Les objets sont *regroupés* dans des classes qui les décrivent de manière abstraite
- La classe intègre les concepts de type et de module

Implémentation en Java/C#

Objets et classes

Bachir Djafri
Lab. IBISC / Dép. Informatique
Université d'Évry Val d'Essonne
bachir.djafri@ibisc.univ-evry.fr
http://www.ibisc.univ-evry.fr/~djafri

Définition de classes (1)

```
public class Point {
   public double x, y; // Coordonnées du point
}
```

```
public class MonProgramme {

   public static void main(String[] args) {

      Point p;     // une référence sur un objet Point et non un objet
      p = new Point();    // p référence un objet alloué en mémoire
      System.out.println("Les coordonnées du point : x="+p.x+"; y="+p.y);
      p.x = 5.3;
      p.y = 13.5;
      System.out.println("Les coordonnées du point : x="+p.x+"; y="+p.y);
   }
}
```

Définition de classes (2)

```
public class Point {

   public double x, y;    // Les coordonnées du point

   // méthode de déplacement
   public void move(double d1, double d2){  // une méthode de déplacement
      x += d1;    y += d2;
   }

}
```

```
public class MonProgramme {
   public static void main(String[] args) {
      Point p;          // une référence sur un objet point, pas un objet
      p = new Point();  // p référence un objet alloué en mémoire
      p.move(3.5, -1.1); // déplacement du point p
      System.out.println("Les coordonnées du point : x=" + p.x + " ; y=" + p.y);
   }
}
```


Définition de classes (3)

```
public class Point {
   public double x, y;    // Les coordonnées du point

   // un constructeur
   public Point(double a, double b) {
      x = a;
      y = b;
   }
   public void move(double d1, double d2){
      x += d1;    y += d2;
   }
}
```

```
public class MonProgramme {

   public static void main(String[] args) {
      Point p;          // une référence sur un objet point, pas un objet
      p = new Point(5.3, 15.7); // p référence un objet alloué en mémoire
      p.move(3.5, -1.1); // déplacement du point p
      System.out.println("Les coordonnées du point : x=" + p.x + " ; y=" + p.y);
   }
}
```


Définition de classes (4)

```
public class Point {
   double x, y;

   public Point(double x, double y) {
      this.x = x;       // this fait référence à l'instance (l'objet)
      this.y = y;       // sur laquelle opère le constructeur
   }
   public Point(Point p) {
      x = p.x; y=p.y;
   }
   public Point() {
      this(0.0, 0.0); // appelle le 1er constructeur
      // this(new Point(1.3, 5.)); // appelle le 2ème constructeur
   }
   ...
}
```


Création d'objets

- Pour manipuler un objet, on déclare une référence sur la classe de cet objet : `Point p;`
- Pour créer un objet, on instancie une classe en appliquant l'opérateur **new** sur un de ses constructeurs. Une nouvelle instance de cette classe est alors allouée en mémoire :

```
p = new Point(5.3, 15.7); // création d'un objet point
```

- Toute classe possède un constructeur par défaut (implicite, sans paramètres) qui peut être redéfini.
- Une classe peut avoir plusieurs constructeurs qui diffèrent par le nombre et le type de leurs paramètres.

Les (Objets) tableaux

- Création et initialisation

```
int[] tableauEntiers = new int[13]; // un tableau de 13 entiers (0..12)
Point[] tableauPoints = new Point[25]; // un tableau de 25 points
char[] chaine = null; // une référence vers un tableau de caractères
```

- Tableaux multidimentionnels

```
double[][] tableauDoubles; // juste un référence
Color[][][] cubeRGB = new Color[256][][];
int[][] T = {{0}, {1,2}, {3,4,5}, {6,7,8,9}, {10,11,12,13,14}};
```

- Manipulation

```
int l = tableauPoints.length; // l = 25
int e = tableauEntiers[45]; // java.lang.ArrayIndexOutOfBoundsException
```


Les objets chaînes de caractères

- Instances de la classe **String**

- Création et initialisation

```
String message = new String("Bonjour");
String message2 = message;
// message2 'pointe' sur le même objet que message
String vide = ""; // chaîne vide
String msg = "Hello!";
String s = msg + message; // s = "Hello! Bonjour"
```

- Manipulation (voir classe String)

```
int l = msg.length(); // l = 6
char c = msg.charAt(4); // c = 'o';
```


Structure des classes (1)

- Une classe est un agrégat **d'attributs** et de **méthodes** : les *membres* de la classe.
- Les méthodes sont définies directement au sein de la classe.
- L'accessibilité des membres d'une classe est pondérée par des critères de visibilité : ***public***, ***private***, …
- Les membres sont accessibles via une instance de la classe (un objet) ou via la classe elle-même (pour les membres *statiques*).

```
p.x = 5.0;           // Accès à l'attribut x de l'instance p
p.move(3.5, -1.1);   // Appel de la méthode move() de l'instance p

double pi = Math.PI; // Accès à l'attribut statique PI de la classe Math
double b = Math.sqrt(3.0); // Appel de la méthode statique sqrt()
                           // de la classe Math
```


Structure des classes (2)

- Les membres statiques (**`static`**) sont partagés par toutes les instances de la classe (objets de la classe).
- Un membre statique peut être accédé soit via une instance de la classe, soit via la classe elle-même.
- Les méthodes statiques ne peuvent pas accéder aux variables d'instances et à **`this`**.
- *Dans certains cas, la classe est désignée par le mot clé* ***`this`***.

Structure des classes (3)

```
public class Cercle {
   public static int nbCercles = 0;
   public static final double PI = 3.1416; // final pour éviter Cercle.PI = 4;
   public double x, y, r; // les cordonnées du centre et un rayon

   public Cercle(double r) { this.r = r; nbCercles++; }

   public Cercle plusGrand(Cercle c) {
      if (c.r > r) return c; else return this; // this fait référence à l'objet
   }                                           // sur lequel opère la méthode

   public static Cercle plusGrand(Cercle c1, Cercle c2) {
      if (c1.r > c2.r) return c1; else return c2;
   }
}
```

```
Cercle c1 = new Cercle(10); Cercle c2 = new Cercle(20);
int n = Cercle.nbCercles;              // n = 2; int n = c1.nbCercles;
Cercle c3 = c1.plusGrand(c2);          // c3 = c2;
Cercle c4 = Cercle.plusGrand(c1, c2); // c4 = c2; Cercle c4 = c3.plusGrand(c1,c2);
```


Structure des classes (4)

- Le mode de passage des paramètres dans les méthodes dépend du type des paramètres :
  - par *référence* pour les objets (copie de références)
  - par *copie* pour les types primitifs

```
public class C {
   void methode1(int i, Point p) {
      i++; p.move(3.0, 3.0);
   }

   void methode2() {
      int i = 0;
      Point p = new Point(5.3, 11.9);
      methode1(i, p);
      System.out.println("i = " + i + ", p.x = " + p.x); // i=0, p.x=8.3
   }
}
```