CENTRALE
L Y O N

JAVA

Principes de fonctionnement

CENTRALE
L Y O N

JAVA : un langage objet (1)

- Une classe est un modèle pour plusieurs objets à particularité similaire. Les classes réunissent toutes les caractéristiques d'un ensemble d'objets donnés.
- Une instance de classe est l'autre nom d'un objet réel. Les classes sont une représentation abstraite d'un objet, une instance en est une représentation concrète.
- Une bibliothèque de classes est un ensemble de classes.
- Les variables d'instance définissent les attributs d'un objet. La classe spécifie le type d'attribut et chaque objet stocke ses propres valeurs pour cet attribut.
- Les méthodes sont des fonctions définies dans les classes. Elles agissent sur les objets de ces dernières.
- Il existe des variables et des méthodes d'instance et de classe.

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

JAVA : un langage objet (2)

Java est proche de C++ et de C, mais :

- boolean est un véritable type de données avec les valeurs True ou False
- La méthode ***main ()*** est la méthode principale exécutée au lancement.
- Avec l'héritage, toutes les classes (celles écrites ou issues d'autres bibliothèques de classes ainsi que les classes utilitaires standard) suivent une hiérarchie stricte.
- **Le sous-classement** concerne la création d'une classe qui hérite d'autres classes dans la hiérarchie. Avec le sous-classement, il suffit de définir les différences entre la classe et ses parents. Les comportements semblables sont tous disponibles pour la classe grâce à l'héritage.
- La classe ***Object*** se trouve au sommet de la hiérarchie des classes Java. Toutes les classes héritent de cette super-classe.
- La définition d'une classe sans super-classe en première ligne amène Java à supposer qu'elle hérite de la classe ***Object***.

BTD/GL/JAVA

3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

JAVA : un langage objet (3)

- Le polymorphisme consiste à créer dans une sous-classe une méthode dont la signature (nom, nombre et type d'arguments) est identique à celle d'une super-classe. La nouvelle méthode " masque " alors celle de la super-classe.
- Java se limite à l'héritage simple : Pour créer une sous-classe on utilise le mot clé ***extends***

```
public class XXX extends java.applet.Applet { }
```

- **Une interface** est une collection de nom de méthodes sans définition. Elle indique qu'une classe a des comportements en plus de ceux hérités des super-classes.
- **Les packages** regroupent des classes et des interfaces. Grâce à eux, les groupes de classes sont disponibles uniquement s'ils sont nécessaires. Ainsi, les conflits éventuels sur les noms de classes de différents groupes sont éliminés.
- Pour référencer une classe d'un package, il faut décrire le chemin complet en utilisant la notation pointée. Seul le package ***Java.lang*** est accessible directement.

BTD/GL/JAVA

4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Termes et concepts (1)

- **Classe :** modèle d'objet qui contient des variables et des méthodes. Celles-ci définissent le comportement et les attributs. Les classes peuvent hériter de variables et de méthodes issues d'autres classes.
- **Objet :** instance concrète d'une classe. Plusieurs objets accèdent aux mêmes méthodes si ce sont des instances de la même classe. Cependant, leurs variables d'instance ont souvent des valeurs différentes.
- **Instance :** représente la même chose qu'un objet. Un objet est une instance d'une classe.
- **Super-classe :** une classe située au-dessus de ses sous-classes dans la hiérarchie de l'héritage.
- **Sous-classe :** une classe au-dessous de sa super-classe dans la hiérarchie de l'héritage. La création d'une nouvelle classe s'appelle sous-classement.

BTD/GL/JAVA 5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Termes et concepts (2)

- **Méthode d'objet :** méthode définie dans une classe qui s'applique à une instance de cette classe. En général, elles sont simplement appelées méthodes.
- **Méthode de classe** : méthode définie dans une classe qui s'applique sur la classe elle-même ou sur un objet.
- **Variable d'instance :** variable qui appartient à un objet donné et dont la valeur est stockée avec l'objet.
- **Variable de classe :** variable qui appartient à la classe et à tous les objets et est stockée dans la classe.
- **Interface :** ensemble de spécifications de comportements abstraits utilisable par une classe donnée.
- **Package :** ensemble de classes et d'interfaces. Dans le cas des classes incluses dans des packages différents de java.lang, le nom du package complet doit être indiqué pour les importer ou les référencer.

BTD/GL/JAVA 6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les bases de Java (1)

- Java ressemble à C++, et par extension, au langage C. Une grande partie de sa syntaxe est connue. On ne donnera ci-après que les différences.
- Java possède trois sortes de variables : les **variables d'instance**, les **variables de classes** et les **variables locales**. Les variables locales suivent les règles de gestion des variables de bloc, elles disparaissent après chaque exécution de la méthode ou du bloc.
- Java ne possède pas de variables globales. Les variables d'instance et de classe les remplacent. Elles servent à communiquer des informations dans et entre les différents objets.

BTD/GL/JAVA 7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les bases de Java (2)

- Les déclarations de variables sont obligatoires, elles peuvent se situer n'importe où dans la méthode mais sont souvent situées en tête.
- L'initialisation peut être faite lors de la déclaration. Les variables locales doivent avoir une valeur avant d'être utilisées, sinon le compilateur signale une erreur. Les variables d'instance et de classe sont initialisées automatiquement (***null*** pour les instances de classe, ***0*** pour les variables numériques,***'\0'*** pour les caractères et ***false*** pour les booléens.
- Java différencie les majuscules et les minuscules.
- Par convention les noms de variables Java sont représentatifs des variables. Souvent composés de plusieurs mots combinés. Le premier écrit en minuscules et les suivants commençant pat une majuscule.

BTD/GL/JAVA 8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Types de variables

- 8 types de base, nom de classe, tableau
- Entiers : **byte** (8 bits : -128 à 127), **short** (16 bits : -32 768 à 32 767) , **int** (32 bits), **long** (64 bits)
- Réels : nombres à virgule flottante **float** (32 bits, simple précision), **double** (64 bit, double précision)
- Charactères : **char** (en codification Unicode non signé avec une précision de 16 bits)
- Logiques : **boolean** avec deux valeurs true ou false

BTD/GL/JAVA 9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

La programmation de base

- **Variables** (identificateur, type, valeur)
- **Les méthodes**
- **Les commentaires :**

```
//
/* vvvvvv */
/** vvvvvvvv */
```

- **Le premier programme :**

```
class Bonjour {
  public static void main(String args[]) {
    System.out.println("Bonjour");
  }
}
```

BTD/GL/JAVA 10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les variables (1)

- **Syntaxe :** *Type identificateur ;*
- Huit types de variables

Nom du type	Taille utilisée	Genre	Valeurs
boolean	8-bits	booléen	"true" ou "false"
byte	8-bits	entier	-128 à +127
short	16-bits	entier	-32768 à +32767
int	32-bits	entier	-2147483648 à 2147483647
long	64-bits	flottant	-9223372036854775808 à 9223372036854775807
float	64-bits	flottant	1.4013e-045 à 3.40282c+038 (en négatif et en positif)
double	64-bits	flottant	2.22507e-308 à 1.79769e+308 (en négatif et en positif)
char	16-bits	caractèretout caractère	

BTD/GL/JAVA

11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les variables (2)

```
class SurfaceRectangle {
  public static void main(String args[]) {
    int largeur;
    int hauteur;
    int surface;
    largeur = 5;
    hauteur = 10;
    surface = largeur * hauteur;
    System.out.println(surface);
  }
}
```

- Visibilité et blocs : règle classique - toute variable est visible dans le bloc où elle est définie et dans ses sous-blocs (sauf si elle y est redéfinie)

BTD/GL/JAVA

12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les variables (3)

- Format des nombres (entiers, décimaux), des caractères et des booléens
  - Nombres entiers :

```
byte    nombrehexa     = Oxff;
int     nombredecimal  = 1996;
long    nombre0ctal    = 01234567L;
long    autrehexa              = OXdeadbeefL;
long    calcul         = 12L * nombre0ctal;
```

  - Nombres décimaux :

```
float pi = 3.1415f , a = -0.3f;
float b = +.lf , c = 5F , d = lel0f;
double v = 3d , w = 1.1;
double x = lel , y = -5.8e-3d , z = .3;
```

BTD/GL/JAVA

13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Caractères spéciaux

Représentation	Signification
'\n'	nouvelle ligne (NL)
'\t'	fabulation (TAB)
'\b'	*backspace (BS)*
'\r'	retour de chariot (CR)
'\f'	fin de page (FF)
'\\'	*backslash* \
'\''	apostrophe
'\»'	guillemet
'\0ddd'	caractère ASCII ddd (octal)
'\xdd'	caractère ASCII dd (hexadécimal)
'\udddd'	caractère Unicode dddd (hexadécimal)

BTD/GL/JAVA

14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Conversion de type

– **Parfois, on déclare une variable d'un certain type et on doit par la suite copier sa valeur dans une autre variable d'un autre type. On dit alors que l'on doit effectuer une conversion de type :**

```
int entier = 5;
float flottant = (float)entier ;
float decimal = 5.6;
int arrondi = (int)decimal;
```

– Attention : une conversion peut entraîner une perte d'information (et de précision)

BTD/GL/JAVA

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les tableaux simples et multidimensionnels

- Les tableaux simples et multidimensionnels :

Déclaration, puis allocation

int tab [] ;

tab = new int[10] ;

ou directement

char echiquier [] = new char [64] ;

char echiquier [] [] = new char [8] [8] ;

Initialisation d'un tableau

int premiers [] = {3, 5, 7, 11, 13} ;

int carre3x3 [][] = {00,01,02}, {10,11,12}, {20,21,22}} ;

BTD/GL/JAVA

16

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les arguments d'une application

```
class AffParam {
  public static void main(String args[]) {
    System.out.println("J'ai recu " + args.length +
   "paramètre(s)");
    System.out.println(args[0]);
    System.out.println(args[1]);
  }
}
```

Le tableau ***args*** contient tous les paramètres passés sur la ligne de commande de l 'application

BTD/GL/JAVA

17

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'incrémentation et opérations

- Incrémentations
  - **x++** **post-incrémentation**
  - **++x** **pré-incrémentation**
  - **x--** **post-décrémentation**
  - **- -x** **pré-décrémentation**

  - **x++** **équivaut à x = x + 1**
  - **x--** **équivaut à x = x - 1**
- Opérations mathématiques
  - **négation -x**
  - **addition x+y**
  - **soustraction x-y**
  - **multiplication x*y**
  - **division x/y**
  - **modulo x%y**

BTD/GL/JAVA

18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations mathématiques et logiques

x += y	équivalent à x = x + y
x -= y	équivalent à x = x - y
x *= y	équivalent à x = x * y
x /= y	équivalent à x = x / y
x % = y	équivalent à x = x % y

- Opérations logiques :

complément (non) !x		et	x & y
ou	x I y	ou-exclusif (xor)	x ^y
décalage à gauche	x << y		
décalage à droite	x >> y (avec extension du signe)		
décalage à droite	x >>> y (avec insertion de 0)		

x &=y	équivalent à x = x & y
x I= y	équivalent à x = x I y
x /\= y	équivalent à x = x ^ y
X <<= y	équivalent à x = x << y
X >>= y	équivalent à x = x >> y
x >>>= y	équivalent à X = X >>> y

BTD/GL/JAVA

19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Instructions de contrôle du flot

- **Exécution conditionnelle – IF**
- **L'opérateur conditionnel - ?**
- **Exécution conditionnelle – SWITCH**
- **La boucle WHILE**
- **La boucle DO**
- **La boucle FOR**
- **L'instruction BREAK**
- **L'instruction CONTINUE**

BTD/GL/JAVA

20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exécution conditionnelle – IF

- **Syntaxes :**

  if (expression) instructionsSiVrai ; else instructionsSiFaux ;

  if (expression) instructionsSiVrai ;

  if (expression) {instructionsSiVrai1 ; instructionsSiVrai2 ; instructionsSiVrai3 ;}
  else {instructionsSiFaux1 ; instructionsSiFaux2 ; instructionsSiFaux3 ;}

BTD/GL/JAVA 21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérateurs

- Opérateurs de comparaison

a == b	vrai Si a est égal à b
a != b	vrai si a est différent de b
a > b	vrai si a est plus grand que b
a < b	vrai si a est plus petit que b
a >= b	vrai si a est plus grand ou égal à b
a <= b	vrai si a est plus petit ou égal b

- Opérateurs logiques

Opération	Signification logique	Explications
!a	NON a	vrai si a est faux
a & b	a ET b	vrai si a et b sont vrais
a && b	a ET b	vrai si a et b sont vrais
a | b	a OU b	vrai si a ou b est vrai
a || b	a OU b	vrai si a ou b est vrai
a ^ b	a OU-exclusif b	vrai si a est vrai et b est faux ou a est faux et b est vrai. Equivalent à a!= b

BTD/GL/JAVA 22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'opérateur conditionnel - ?

- Il existe un opérateur spécial, qui permet de faire des tests sans l'aide de l'instruction ***if*** :
- **Syntaxe :**

  cond ? siVrai : siFaux

Exemple :

```
int a = 1, b = 5;
int min = a < b ? a : b ;
int max = a < b ? b : a ;
```

BTD/GL/JAVA

23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exécution conditionnelle – SWITCH

- Syntaxe :

```
switch (variable) {
case cas1 : instr11 ; instr12 ; break ;
case cas2 : instr21 ; instr22 ;
case cas3 : instr31 ; instr32 ; break ;
default : instrDefaut ;
} suite ;
```

- On ne peut utiliser SWITCH qu'avec des types primitifs d'une taille maximum de 32 bits (byte, short, int, char) et pas avec des objets complexes, tels que des chaînes de caractères (string) ou des nombres à virgule flottante (float), etc.

BTD/GL/JAVA

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

La boucle WHILE

- Syntaxe :

  while (condition) instruction ;

  Attention pas de ; après la condition sinon c'est considéré comme une instruction vide. Exemple : *while (nombre < aAtteindre)* **;** *nombre++ ;*

  (**si ;** après la condition, on a soit bouclage, soit une fois)

```
class BoucleWhile {
  public static void main(String args[]) {
    int borne = 3400;
    int diviseur = 133;
System.out.println("cherchons le premier multiple de
  "+diviseur+" plus grand ou egal a " + borne);
int fact = borne;
    while ((fact % diviseur) != 0) { fact++;}
    System.out.println("c'est "+fact);
  }
}
```

BTD/GL/JAVA

25

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

La boucle DO

- Syntaxe :

  do instr ; while (condition) ;

```
class BoucleDo {
  public static void main(String args[]) {
    int borne = 3458;
    int diviseur = 133;
System.out.println("cherchons le premier multiple de
  "+diviseur+" plus grand que "+borne);
    int fact = borne;
    do
      fact++;
    while ((fact % diviseur) != 0);
    System.out.println("c'est "+fact);
 }
}
```

BTD/GL/JAVA

26

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

La boucle FOR

- Syntaxes :

  for (expression1 ; expression2 ; expressions3) instruction ;

  for (init ; condition ; incrémentation) instruction ;

  for (; ;) { Instructions ; }
  est équivalent à : *while (true) { instructions ;}*

```
class BoucleFor {
  public static void main(String args[]) {
    for (int i = 1;i <= 10;i++)
      System.out.println(i);
  }
}
```

BTD/GL/JAVA 27

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'instruction BREAK

- L'instruction BREAK sert à rompre le flot d'instructions et à sortir du bloc courant, sans exécuter les instructions suivant le BREAK. Pour sortir de deux boucles imbriquées on utilise un BREAK avec le nom de la boucle que l'instruction BREAK doit quitter (les boucles peuvent être nommées *nom :*).

```
int X, y;
boolean trouve = false ;
ext: for (x = 0; x<tab.length; x++) {
  for (y = 0; y < tab[0].length; y++) {
      if (tab[x][y] == recherche) {
        somme += tab[i];
        trouve = true;
          break ext;
       }
      }
  }
  if (trouve) …;
```

BTD/GL/JAVA 28

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'instruction CONTINUE

- L'instruction CONTINUE permet de sauter les instructions restantes de la boucle pour réévaluer immédiatement la condition.

```
class BreakContinue {
  public static void main(String args[]) {
    int tab[] = {14, 25, 22, 12, 91, 80, 63, 37, 71};
    int somme = 0; int neCorrespondPas = 0;
    int trouves = 0;
    for (int i = 0;i < tab.length;i++) {
      if (tab[i] > 25) {somme += tab[i]; trouves++;
        if (trouves == 3)break; else continue;}
      neCorrespondPas++; }
    System.out.println(neCorrespondPas + " éléments ne répondaient pas
    à la condition Supérieur … 25");
    if (trouves == 3) System.out.println("la moyenne des 3 premiers
    éléments plus grand que 25 est de " + ((float)somme) /
    ((float)trouves));
    else System.out.println("nous n'avons pas trouve 3 éléments plus
    grand que 25 dans le tableau");
  }
}
```

BTD/GL/JAVA 29

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation Objet

- **Les Classes - exemple**

```
class Crayon {
  int longueur = 50;
  float diametre = 5f;
Crayon() {}
  Crayon(float d) {
    diametre = d; }
  Crayon(int l,float d) {
    longueur = l;
    diametre = d;}
  void affiche() {
    System.out.println("longueur = "+longueur);
    System.out.println("diametre = "+diametre); }
  float volume() {
    float r = diametre / 2;
    return 3.14f * r * r * longueur; }
  void plusLong(int enPlus) {
    longueur += enPlus; }
  void plusGros(float enPlus) {
    diametre += enPlus;}
}
```

BTD/GL/JAVA 30

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les méthodes d'instance

- Syntaxe :

  Déclaration :

  type nom (parametre1, parametre2,…) {
  code;
  }

  Exemple : float volume (){ float r=diametre / 2; return 3.14*r *r*longueur; }
  Si type void pas de return

  Appel :

  objet.methode (parametre1, parametre2,…);

- Appeler une méthode uniquement par son nom conduit à appeler la méthode de l'instance dans laquelle on se trouve.
- Appeler une méthode en la précédant par le nom de l'objet, conduit à appeler la méthode de cet objet.

BTD/GL/JAVA 31

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

La surcharge de méthodes

- **Surcharge : déclarer plusieurs méthodes portant le même nom, mais prenant des paramètres différents. Au moment de la génération de code, le compilateur identifie les méthodes à évoquer en s'appuyant sur le nom, le nombre et les types de paramètres (pas le paramètre de retour)**
- **Accès aux variables d'instance**
- On peut accéder aux variables d'instance par la notation *nomObjet*.*nomVariable*. Cette pratique n'est toutefois pas recommandée, car le principe d'encapsulation ne serait plus respecté.
- Il est conseillé d'utiliser uniquement les méthodes offertes par la classe.

BTD/GL/JAVA 32

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les constructeurs

- Un constructeur est une méthode particulière qui n'est exécutée qu'une fois lors de l'instanciation d'un objet ;
- Les constructeurs servent à initialiser les variables contenues dans les objets et portent le même nom que la méthode.
- Le constructeur n'est pas typé.

```
class Crayon { int longueur = 50; float diametre = 5f;
Crayon() { }
  Crayon(float d) { diametre = d;}
  Crayon(int l,float d) {longueur = l;diametre = d;}
  void affiche() {
    System.out.println("longueur = "+longueur);
    System.out.println("diametre = "+diametre); }
  float volume() { float r = diametre / 2;
    return 3.14f * r * r * longueur; }
  void plusLong(int enPlus) {
    longueur += enPlus;}
  void plusGros(float enPlus) {
    diametre += enPlus; }
}
```

BTD/GL/JAVA

33

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les constructeurs (suite)

```
class ConstructeursCrayon {
  public static void main(String args[]) {
    Crayon crayon1 = new Crayon();
    Crayon crayon2 = new Crayon(8.5f);
    Crayon crayon3 = new Crayon(33, 10.2f);
    System.out.println("crayon 1:");
    crayon1.affiche();
    System.out.println("crayon 2:");
    crayon2.affiche();
    System.out.println("crayon 3:");
    crayon3.affiche();
  }
}
```

BTD/GL/JAVA

34

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les références et la comparaison d'objets

- Les objets que l'on déclare ne contiennent pas un objet mais une référence à un objet.
- c2 = c1 // on copie dans c2 la référence à l'objet contenu dans c1.
- L'opérateur = = compare les références.
- Si l'on veut comparer l'égalité des variables de ces deux instances, il faut munir la classe d'une méthode à cet effet :

```
class Rectangle { int longueur; int largeur;
   Rectangle(int longueur,int largeur) {
     this.longueur = longueur;
     this.largeur = largeur;}
   boolean estEgalA(Rectangle r) {
     return (r.longueur == longueur && r.largeur ==
    largeur); }
}
```

BTD/GL/JAVA

35

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Comparaisons (suite)

```
class CompObjets {
  public static void main(String args[]) {
    Rectangle r1 = new Rectangle(10,15);
    Rectangle r2 = new Rectangle(10,15);
    if (r1.estEgalA(r2))
      System.out.println("r1 et r2 sont ,égaux");
    else
      System.out.println("r1 et r2 sont différents");
  }
}
```

BTD/GL/JAVA

36

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'objet this

- Référence à soi-même
- l'objet this est égal à l'instance de l'objet dans lequel il est utilisé

BTD/GL/JAVA 37

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'héritage (1)

- On peut étendre une classe sans avoir à la réécrire complètement :

  Héritage exprime : " hérite de ", " est dérivée de ", " est une sous-classe de ", " est une super-classe, surclasse ou classe parente de "
- **Syntaxe :** *extends*

```
class Voiture {
  int noCarteGrise;
  int immatriculation;
  Voiture(int noCarteGrise,int immatriculation) {
    this.noCarteGrise = noCarteGrise;
    this.immatriculation = immatriculation; }
  void afficheCarteGrise() {
    System.out.println("carte grise: "+noCarteGrise);  }
  void afficheImmatriculation() {
    System.out.println("immatriculation: "+immatriculation);}
  void affiche() {
    afficheCarteGrise();
    afficheImmatriculation();
  }
}
```

BTD/GL/JAVA 38

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'héritage (2)

```
class VoitureCouleur extends Voiture {
   String couleur;
   VoitureCouleur(int carteGrise,int immatr,String couleur) {
     super(carteGrise,immatr);
     this.couleur = couleur;
   }
   void afficheCouleur() {
     System.out.println("couleur: "+couleur);
   }
   void affiche() {
     super.affiche();
     afficheCouleur();
   }
 }
```

BTD/GL/JAVA

39

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'héritage (3)

```
class Route {
  public static void main(String args[]) {
    Voiture buggy = new Voiture(1234,102313);
    VoitureCouleur buggyRouge = new
   VoitureCouleur(1234,102313,"rouge");

    System.out.println("La buggy");
    buggy.afficheCarteGrise();
    buggy.afficheImmatriculation();

    System.out.println("La buggy rouge");
    buggyRouge.afficheCarteGrise();
    buggyRouge.afficheImmatriculation();
    buggyRouge.afficheCouleur();

    System.out.println("Tout sur la buggy");
    buggy.affiche();
    System.out.println("Tout sur la buggy rouge");
    buggyRouge.affiche();
  }
}
```

BTD/GL/JAVA

40

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'héritage (4)

- Les méthodes et les constructeurs de Voiture :

```
Voiture(int noCarteGrise,int immatriculation)
void afficheCarteGrise()
void afficheImmatriculation()
void affiche()
```

- Les méthodes et les constructeurs de VoitureCouleur :

```
VoitureCouleur(int carteGrise,int immatr,String couleur)
void afficheCouleur()
void affiche()
void afficheCarteGrise() (hérité)
void afficheImmatriculation() (hérité)
```

Les constructeurs ne sont pas hérités

BTD/GL/JAVA

41

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

L'opérateur instanceof

- Pour déterminer la classe de l'objet concerné

objet instanceof classe

- Pour pouvoir utiliser la méthode de la classe il faut effectuer la conversion " casting " :

```
void objectInfo(Object o) {
    if (o instanceof Voiture) {
            Voiture v = (Voiture)o;
            v.afficheImmatriculation();
    }
}
```

BTD/GL/JAVA

42

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les interfaces (1)

- Une interface est une sorte de standard auquel une classe peut choisir de répondre. On dit également que c'est un modèle d'implémentation.
- Tous les objets qui se conforment à cette interface, qui implémentent une interface, possèdent les méthodes déclarées dans celle-ci.
- Exemple Document :

```
interface Document {
  public String titre();
  public String auteur();
  public String editeur();
  public java.util.Date dateDeParution();
}
```

- Toute classe qui se conforme à l'interface Document possède au moins les méthodes *titre(), auteur()* et *editeur()* qui ne prennent pas de paramètres et qui retournent un objet de la classe *String*. Elle possède également la méthode *dateDeParution()* qui retourne un objet de la classe *java.util.Date*.
- **En revanche, rien n'est dit sur la fonction de ces méthodes.**

BTD/GL/JAVA

43

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les interfaces (2)

```
public class Livre implements Document {
String Titre; String Auteur; String Editeur;
  String lignes[] = new String[1000];
  // on prévoit 1000 lignes au maximum
  int nombreLignes = 0;
  int lignesParPage = 66;
  java.util.Date laDateDeParution;

  public Livre(String titre, String auteur, String editeur) {
    this.Titre = titre;
    this.Auteur = auteur;
    this.Editeur = editeur;
    // crée un nouvel objet date qui contient
    // la date d'aujourd'hui
    laDateDeParution = new java.util.Date();
  }
  public String titre() { return Titre;}
  public String auteur() { return Auteur; }
  public String editeur() { return Editeur;}
  public void afficheLigne(int no) {
    System.out.println(lignes[no]);
  }
```

BTD/GL/JAVA

44

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les interfaces (3)

```
    public java.util.Date dateDeParution() {
      return laDateDeParution;
    }
    public int nombreDePages() {
      return nombreLignes / lignesParPage;
    }
    public void ajouteTexte(String texte) {
      lignes[nombreLignes++] = texte;
    }
  }
```

- Les méthodes *afficheLigne(int no)*, *nombreDePages ()* et *ajouteTexte(String texte)* ne font pas partie de l'interface.

BTD/GL/JAVA

45

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les interfaces (4)

- **Un autre exemple classe Photo**

```
public class Photo implements Document {
  java.awt.Image laPhoto;
  java.util.Date laDateDeParution;
  boolean couleur; String Titre; String Auteur;  String Editeur;

  public Photo(String titre, String auteur, String editeur, boolean
   couleur, java.awt.Image laPhoto) {
    this.Titre = titre;
    this.Auteur = auteur;
    this.Editeur = editeur;
    this.couleur = couleur;
    this.laPhoto = laPhoto;
    laDateDeParution = new java.util.Date();
}
  public String titre() { return Titre; }
  public String auteur() { return Auteur; }
  public String editeur() { return Editeur; }
  public java.util.Date dateDeParution() {return laDateDeParution; }
  public java.awt.Image image() { return laPhoto; }
  public boolean estEnCouleur() { return couleur; }
}
```

BTD/GL/JAVA

46

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les interfaces (5)

```
// la bibliothèque peut stocker 100 documents
  Document docs[] = new Document[100];
  int nombre;
public void ajoute(Document d) {
    docs[nombre++] = d;}
  public Document rechercheParTitre(String titre) {
    for (int i = 0; i < nombre; i++) {
      String titreDuDocument = docs[i].titre();
      // vrai si les titres sont égaux
      if (titreDuDocument.equals(titre))
        return docs[i];}
    return null;}
}
class GestionBiblio {
  public static void main(String args[]) {
    Livre JavaFacile = new Livre("Java Facile", "Alexandre Maret",
   "Marabout");
    Livre Autocad = new Livre("Autocad 13", "JP Couwenbergh",
   "Marabout");
//    Photo paysage = new Photo("Genève - Paysage", "Toto", "Belles
   Images", true, chargeImage("paysage.gif"));
    Biblio principale = new Biblio();
```

BTD/GL/JAVA

47

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les interfaces (6)

```
 JavaFacile.ajouteTexte("Bonjour");
    JavaFacile.ajouteTexte("Grâce … ce livre vous");
    JavaFacile.ajouteTexte("allez apprendre Java");
principale.ajoute(JavaFacile);
    principale.ajoute(Autocad);
//    principale.ajoute(paysage);
Document retrouve = principale.rechercheParTitre("Java Facile");
    if (retrouve == null)
      System.out.println("impossible de trouver le document");
    else { System.out.println("l'auteur est : "+retrouve.auteur());
      if (retrouve instanceof Livre) {
        System.out.println("il s'agit d'un livre");
        ((Livre)retrouve).afficheLigne(2);
      } else if (retrouve instanceof Photo) {
        System.out.println("il s'agit d'une photo");
        if (((Photo)retrouve).estEnCouleur())
          System.out.println("elle est en couleurs");
        else  System.out.println("elle est en noir et blanc");
      } else
        System.out.println("il s'agit d'un document de type inconnu");
   }} }
```

BTD/GL/JAVA

48

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Une interface peut hériter d'une autre interface

- Une interface peut également hériter d 'une autre interface

```
interface DocumentOfficiel
extends Document {
String approuvePar ();
int reference ();
int niveauClassification ();
}
```

- Toute classe implémentant l'interface DocumentOfficiel est également conforme à l'interface Document.

BTD/GL/JAVA

49

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les objets dans les paramètres de méthodes

- La référence de l'objet est passée par valeur. Lorsque l'on passe un objet par paramètre, il est possible de modifier cet objet en utilisant ces méthodes. Par contre, il n'est pas possible de remplacer cet objet par une nouvelle instance. Le changement n'aura lieu que localement à la méthode.
- **L'objet null**
- L'objet null n'appartient pas à une classe, mais il peut être utilisé en lieu et place d'un objet de toute classe, dans un paramètre. Attention aux problèmes d'exécution, où il faut travailler avec des bons objets.
- Il faut donc d'abord tester l'existence.

BTD/GL/JAVA

50

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les paquetages

- Pour ranger une classe dans un paquetage :
  Syntaxe : *package nomDuPaquetage ;*
- **Une hiérarchie des paquetages est exprimée par la notation pointée.**
- La hiérarchie des paquetages apparaît également au niveau des répertoires.
  - **Utilisation d'un paquetage**
- On donne le nom complet pour une classe (en notation pointée).
- On exprime l'importation de classes utilisées par :
- *Import nomDuPaquetage.* ;*
  - **Collision de classes**
- Pour éviter la collision de noms de classes, il faut donner les noms complets.

BTD/GL/JAVA

51

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les paquetages standard

- Classes standard : java.lang
- Classes très utiles : java.util
- Gestion d'applets : java.applet
- Interface graphique : java.awt
- Gestion des images : java.awt.image
- Interface avec le système d'exploitation : java.awt.peer
- Entrées-sorties fichiers : java.io
- Accès au réseau : java.net

BTD/GL/JAVA

52

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les modificateurs (1)

- **Visibilité des entités – public, private, protected**
- Public : visible
- Private : visible seulement dans la classe où elle définie
- Protected : même paquetage et ses sous- classes
- Les modificateurs ne concernent pas les variables locales, mais les variables d'instance et de classe.

BTD/GL/JAVA

53

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les modificateurs (2)

5 niveaux de protection différents :

- **public** : variable, méthode ou classe sera visible par toutes les autres méthodes, à l'intérieur ou à l'extérieur de l'objet. (généralement on évite).
- **.protected** : méthode ou variable accessible par les méthodes présentes dans le même paquetage (ne s'applique pas à la classe).
- **.**par défaut, **package, friendly** : classe, méthode, variable visible par toutes les classes se trouvant dans le même paquetage.
- **.private protected** : méthode ou variable qui n'est visible que dans sa classe ou dans ses sous -classes (ne s'applique pas à la classe)
- **private : seules les méthodes de la classe dans laquelle on définit une entité *private* pourront la voir.**

BTD/GL/JAVA

54

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les modificateurs (3)

Static : variable de classe

- Une méthode statique (static) est une méthode qui n'agit pas sur des variables d'instance, mais uniquement sur des variables de classe.
- On parle de méthode de classe.
- Il n'est pas possible d'appeler une méthode d'instance ou d'accéder à une variable d'instance à partir d'une méthode de classe.
- **Final** : constante - Une variable *final* est constante.
- Une méthode *final* ne peut pas être remplacée dans une sous-classe. Elle peut être optimisée par le compilateur (car elle ne sera pas modifiée).
- Le modificateur *final* ajouté à une classe interdit de créer une classe qui en hérite.

BTD/GL/JAVA

55

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les modificateurs (4)

- **Abstract :** retardé, pas d'instanciation possible, seulement un moule partiel
- **Synchronized** :accès concurrent
- **Volatile** : pas de stockage dans un registre, la valeur est réécrite si nécessaire.
- **Native** : dans un autre langage

BTD/GL/JAVA

56

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les exceptions

- **Pour traiter des erreurs :**
  - **déclarer une exception : throws**
  - **lever une exception : throw**
  - **attraper une exception : try et catch**
- **classe Exception**
- **objet exception : instance d 'une classe dérivée de la classe Throwable**

BTD/GL/JAVA

57

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les exceptions exemple (1)

```
class divisionParZeroException extends Exception {}
class operationInconnueException extends Exception {
  public operationInconnueException(String s) {
    super(s);
  }}
class divMul {
private static int execOperation(int operande1, String
  operation, int operande2)
  throws divisionParZeroException,
  operationInconnueException {
    if ("*".equals(operation))
      return operande1*operande2;
    else if ("/".equals(operation)) {
      if (operande2 == 0)
        throw new divisionParZeroException();
      return operande1/operande2;
    } else
      throw new operationInconnueException(operation);
  }
```

BTD/GL/JAVA

58

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les exceptions exemple (2)

```
  public static void main (String args[]) {
     System.out.println("exemple de traitement d'exceptions");
     int    operande1[] = {3,  56, 33, 25};
     String operation[] = {"*","/","+","/"};
     int    operande2[] = {5,  4,  5,  0};
for (int n = 0;n < operande1.length;n++) {
      try {
        System.out.println("Calcul de " + operande1[n] +
  operation[n] + operande2[n]);
        System.out.println("Resultat = " +
  execOperation(operande1[n], operation[n], operande2[n]));
     } catch (divisionParZeroException e) {
        System.out.println("Division par zero.");
     } catch (operationInconnueException e) {
        System.out.println("Operation inconnue : " +
  e.getMessage());
     } catch (Exception e) {
        System.out.println("une autre exception est survenue.");
     } finally {
        System.out.println("clause finally appellee");
     }  } }}
```


CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Threads

- Mise en œuvre du parallélisme
- Les processus légers partagent le code, les données et l ’espace d ’adressage
- Deux approches :
  - par héritage de la classe Thread
  - par interface runnable

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Par héritage de la classe Thread

- on doit fournir une méthode run()
- lancement par la méthode start() qui appelle automatiquement la méthode run()
- la méthode run() travaille souvent en boucle infinie
- join() pour attendre la fin de tous les threads pour terminer l 'exécution

BTD/GL/JAVA 61

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Thread - exemple héritage

```
class afficheur extends Thread {
  /** constructeur permettant de nommer le processus */
  public afficheur(String s) {
    super(s);
  }
  /** affiche son nom puis passe le controle au suivant */
  public void run() {
    while (true) {
      System.out.println("je suis le processus "+getName());
      Thread.yield();    // passe le controle
    }
  }
}
class thread12 {
  public static void main(String args[]) {
    afficheur thread1 = new afficheur("1");
    afficheur thread2 = new afficheur("2");
    thread1.start();
    thread2.start();
    while (true) {
      System.out.println("je suis la tache principale !");
      Thread.yield();
    }
  }
}
```

BTD/GL/JAVA 62

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Par interface runnable

La classe implémente l 'interface runnable

```
class afficheurRunnable implements Runnable {
  boolean cont = true;
  String nomProcessus;
  Thread th;
  /** constructeur permettant de nommer le processus */
  public afficheurRunnable(String s) {nomProcessus = s;}
  public void start() {
    if (th == null) {
      th = new Thread(this,nomProcessus);
      th.start();}}
  public void stop() {
    if (th != null) {
      th.stop();
      th = null;}}
  /** affiche son nom puis passe le controle au suivant */
  public void run() {
    while (cont) {
      System.out.println("je suis le processus "+th.getName());
      Thread.yield();    // passe le controle
    }
  }
}
```

BTD/GL/JAVA

63

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Thread - exemple interface

```
class runnable12 {
  public static void main(String args[]) {
    afficheurRunnable run1 = new afficheurRunnable("1");
    afficheurRunnable run2 = new afficheurRunnable("2");
    run1.start();
    run2.start();
    while (true) {
      System.out.println("je suis la tache principale !");
      try {
        Thread.sleep(20);
      } catch (InterruptedException e) {}
    }
  }
}
```

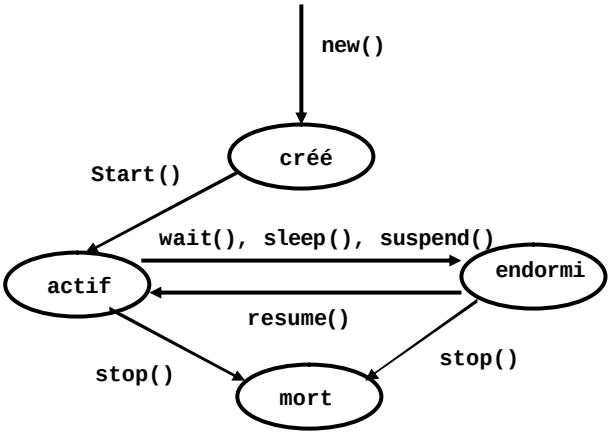
BTD/GL/JAVA

64

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Etats d 'un thread



BTD/GL/JAVA

65

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Opérations sur les threads

- new() pour créer une instance
- start() pour démarrer : par appel automatique à la méthode run()
- pour endormir sleep(), wait(), suspend()
- sleep() un certain temps
- wait() en attente d 'une condition
- suspend() pour être réveillé par resume()
- stop() pour finir définitivement
- isAlive() pour savoir si en thread est vivant

BTD/GL/JAVA

66

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exclusion mutuelle

- Pour partager des données sans problème il faut instaurer une séquence critique : utiliser une méthode synchronized
- le verrouillage n 'est effectif que pour les méthodes synchronized sur la même instance
- on peut appliquer synchronized sur une méthode statique (de classe) l 'exclusivité porte alors sur toute la classe (pour cette méthode)
- on peut rendre critique une instance, en lui appliquant synchronized (objet)

BTD/GL/JAVA

67

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exclusion exemple

Sécurisation d'une méthode

```
class mutexStatic {
  private int accumulateur = 0;
  private static int acces = 0;

  public synchronized void stocke(int val) {
    accumulateur += val;
    synchronized (getClass()) {
      acces += 1;
    }
  }

  public int lit() {
    synchronized (getClass()) {
      acces += 1;
    }
    return accumulateur;
  }

  public int acces() {
    return acces;
  }
}
```

BTD/GL/JAVA

68

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Coopération entre threads

- wait() dans une méthode d 'un objet met le thread en attente d 'un signal de réveil
- appel à notify() dans une méthode du même objet réveille le thread en attente
- le méthodes contenant wait() et notify() doivent être synchronized

BTD/GL/JAVA

69

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple tampon actif (1)

```
class tamponCirc {
private Object tampon[];
  private int taille;
  private int en, hors, nMess;
/** constructeur. crée un tampon de taille éléments */
  public tamponCirc (int taille) {
    tampon = new Object[taille];
    this.taille = taille;
    en = 0;
    hors = 0;
    nMess = 0;
  }
public synchronized void depose(Object obj) {
    while (nMess == taille) {    // si plein
      try {
        wait();                // attends non-plein
      } catch (InterruptedException e) {}
    }
    tampon[en] = obj;
    nMess++;
    en = (en + 1) % taille;
    notify();                // envoie un signal non-vide
  }
```

BTD/GL/JAVA

70

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple tampon actif (2)

```
public synchronized Object preleve() {
    while (nMess == 0) {    // si vide
      try {
        wait();            // attend non-vide
      } catch (InterruptedException e) {}
    }
    Object obj = tampon[hors];
    tampon[hors] = null;    // supprime la ref a l'objet
    nMess--;
    hors = (hors + 1) % taille;
    notify();                // envoie un signal non-plein
    return obj;  }}
class producteur extends Thread {
  private tamponCirc tampon;
  private int val = 0;
  public producteur(tamponCirc tampon) {
    this.tampon = tampon; }
  public void run() {
    while (true) {
      System.out.println("je depose "+val);
      tampon.depose(new Integer(val++));
      try {
        Thread.sleep((int)(Math.random()*100));
        // attente de max 100 ms
      } catch (InterruptedException e) {} }  }}
```


CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple tampon actif (3)

```
class consommateur extends Thread {
  private tamponCirc tampon;
  public consommateur(tamponCirc tampon) {
    this.tampon = tampon;}
  public void run() {
    while (true) {
      System.out.println("je preleve " +
   ((Integer)tampon.preleve()).toString());
      try {
        Thread.sleep((int)(Math.random()*200));
        // attente de max 200 ms
      } catch (InterruptedException e) {}
    }}}
class utiliseTampon {
  public static void main(String args[]) {
    tamponCirc tampon = new tamponCirc(5);
    producteur prod = new producteur(tampon);
    consommateur cons = new consommateur(tampon);

    prod.start();
    cons.start();
    try {
      Thread.sleep(30000); // s'execute pendant 30 secondes
    } catch (InterruptedException e) {}
  }}
```

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les classes standard

BTD/GL/JAVA

73

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

VisualAge for Java

- **Symboles de l'IDE de VisualAge pour Java**
  - **Lorsque vous placez le pointeur de la souris sur un symbole de l'environnement IDE, vous obtenez une aide en incrustation ainsi qu'une description brève dans la ligne d'état. Cependant, cette aide n'est pas disponible pour les symboles suivants :**
  - **Eléments de programme**

    projet
    package
    classe
    interface

BTD/GL/JAVA

74

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Applet JAVA

- Une applet est un programme dynamique et interactif qui s'exécute dans une page Web affichée par un programme de navigation compatible Java comme HotJava ou Netscape.
- Un programme indépendant des plates-formes est portable d'un système à un autre.
- Le pseudo-code est un ensemble d'instructions proches du code machine, mais non spécifique à un ordinateur.
- Le compilateur (***javac***) génère du pseudo-code (fichiers ***.class***) à partir de fichiers source (fichier ***.java***). L'interpréteur (***java***) exécute ces fichiers ***.class***.
- L'intégration de l'applet dans le code HTML :
  <APPLET CODE="XXX.class" WIDTH=150 HEIGHT=25>

BTD/GL/JAVA

75