
Cours Java_I

Cnam Paris

jean-michel Douin, douin@cnam.fr
Nouvelle version : 4 Septembre 2005

Notes de cours java : le langage : Introduction, approche impérative

Les notes et les Travaux Pratiques sont disponibles en
http://jfod.cnam.fr/tp_cdi/douin/

Ce support ne représente que des notes de cours, il ne peut se substituer à la lecture d'un ouvrage sur ce thème; Le livre de Liskov cité en bibliographie en est un exemple.

Sommaire

- **Les objectifs des concepteurs**
- **Présentation des concepts de l'orienté Objet**
- **Types primitifs**
- **Opérateurs**
- **Instructions**
- **La gestion des exceptions**
- **Classe : syntaxe et introduction**
- **Notions de Package**
- **Résumé**

Bibliographie utilisée

- The Java Handbook, Patrick Naughton. Osborne McGraw-Hill. 1996.
<http://www.osborne.com>
- <http://java.sun.com/docs/books/jls/>
- <http://java.sun.com/docs/books/tutorial.html>
- Program Development in Java,
Abstraction, Specification, and Object-Oriented Design, B.Liskov avec J. Guttag
voir <http://www.awl.com/cseng/> Addison Wesley 2000. ISBN 0-201-65768-6

Java : les objectifs

- **« Simple »**
syntaxe " C "
- **« sûr »**
pas de pointeurs, vérification du code à l'exécution et des accès réseau et/ou fichiers
- **Orienté Objet**
(et seulement !), pas de variables ni de fonctions globales, types primitifs et objet
- **Robuste**
ramasse miettes, fortement typé, gestion des exceptions
- **Indépendant d'une architecture**
Portabilité assurée par la présence d'un interpréteur de bytecode sur chaque machine
- **Environnement riche**
Classes pour l'accès Internet
classes standard complètes
fonctions graphiques évoluées
- **Support d'une méthodologie de conception basée sur les « Patterns »**
Conception Orientée Objet

Simple : syntaxe apparentée C,C++

```
public class Num{  
  
    public static int max( int x, int y){  
        int max = y;  
        if(x > y)  
            max = x;  
  
        return max;  
    }  
}
```

Sûr par l'absence de pointeurs accessibles au programmeur

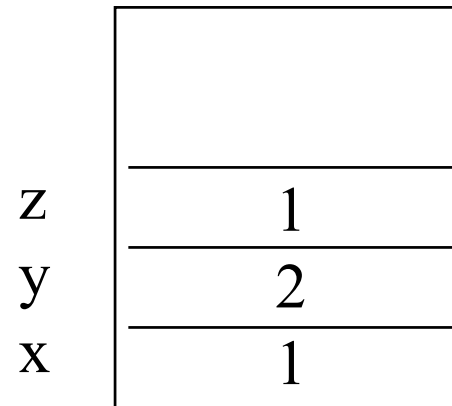
- **Deux types : primitif ou Object** (et tous ses dérivés)

- **primitif :**

```
int x = 1;
```

```
int y = 2;
```

```
int z = x;
```

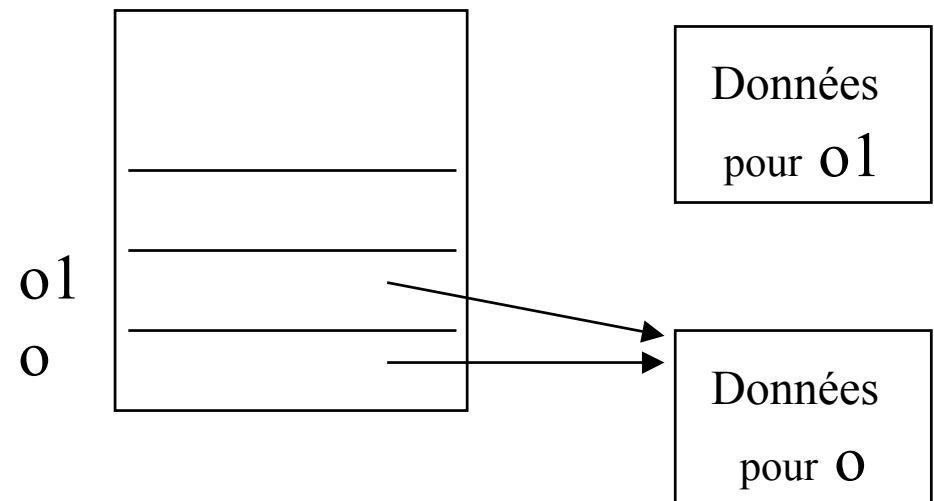


- **Object**

```
Object o = new Object();
```

```
Object o1 = new Object();
```

```
Object o1 = o;
```



Robuste

- **Ramasse miettes ou gestionnaire de la mémoire**
Contrairement à l'allocation des objets, Leur dé-allocation n'est pas à la charge du programmeur
(Ces dé-allocations interviennent selon la stratégie du gestionnaire)
- **Fortement typé**
Pas d'erreur à l'exécution due à une erreur de type
- **Exceptions**
Mécanisme de traitements des erreurs,
Une application ne devrait pas s'arrêter à la suite d'une erreur, (ou toutes les erreurs possibles devraient être prises en compte ...)

Portable

Le source Java
Num.java

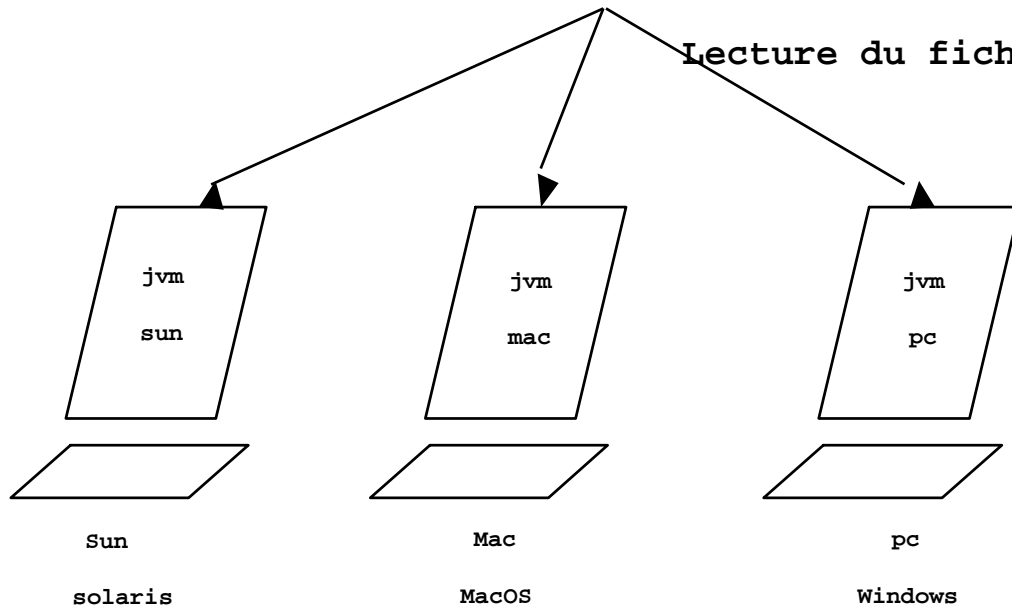
```
public class Num {  
...  
}
```

1) compilation

Le fichier compilé
Num.class

```
1100 1010 1111 1110 1011 1010 1011 1110  
0000 0011 0001 1101 .....
```

Lecture du fichier locale ou distante



2) interprétation

Environnement (très) riche

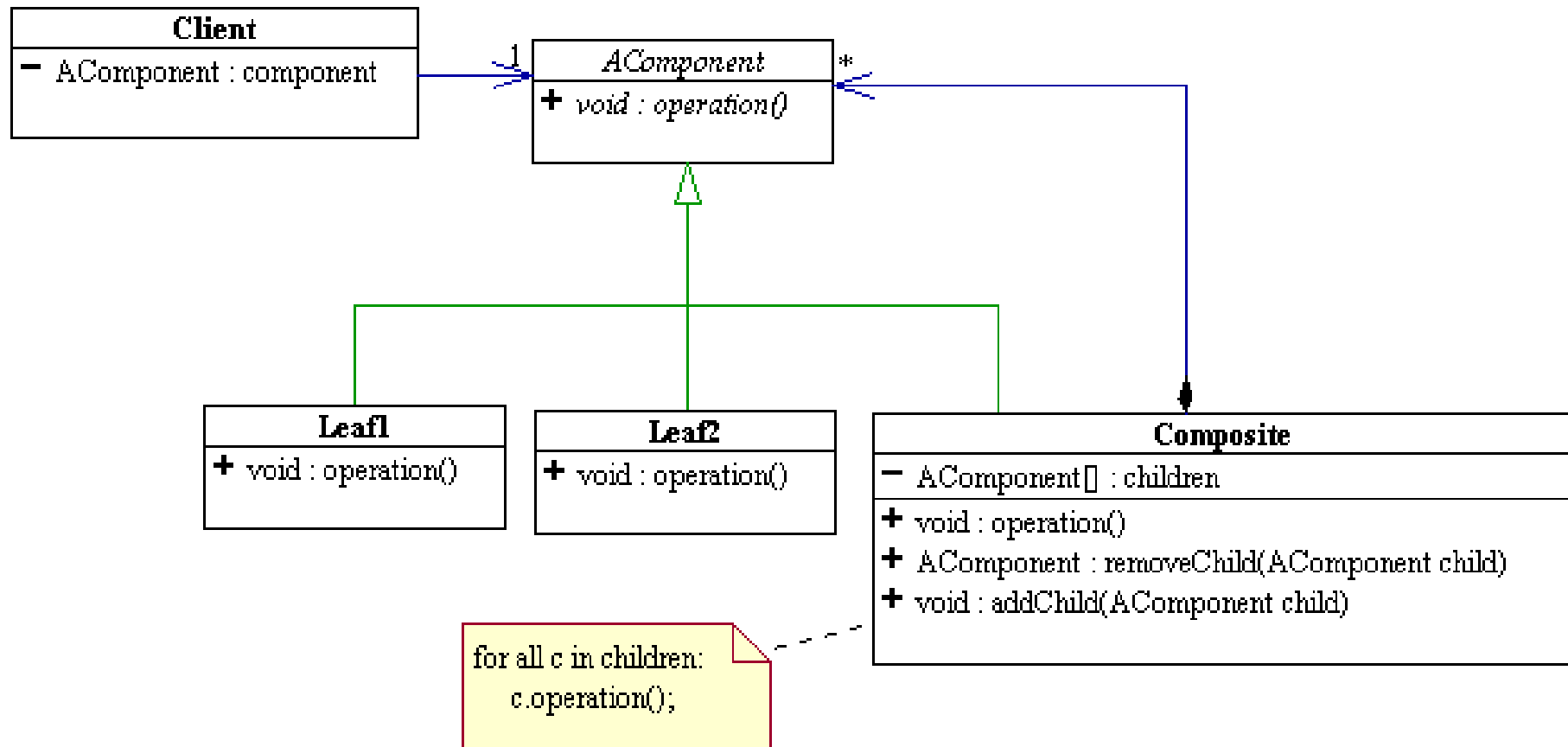
- **java.applet**
 - **java.awt**
 - **java.beans**
 - **java.io**
 - **java.lang**
 - **java.math**
 - **java.net**
 - **java.rmi**
 - **java.security**
 - **java.sql**
 - **java.text**
 - **java.util**
 - **javax.accessibility**
 - **javax.swing**
 - **org.omg.CORBA**
 - **org.omg.CosNaming**
- **Liste des principaux paquetages de la plate-forme JDK 1.2 soit environ 1500 classes !!! Et bien d'autres A.P.I. JSDK, JINI, ...**
- **le JDK1.3/1850 classes,**
 - **le JDK 1.4/2700 classes**
 - **Le JDK1.5 ou j2SE5.0 3260 classes**

Pattern

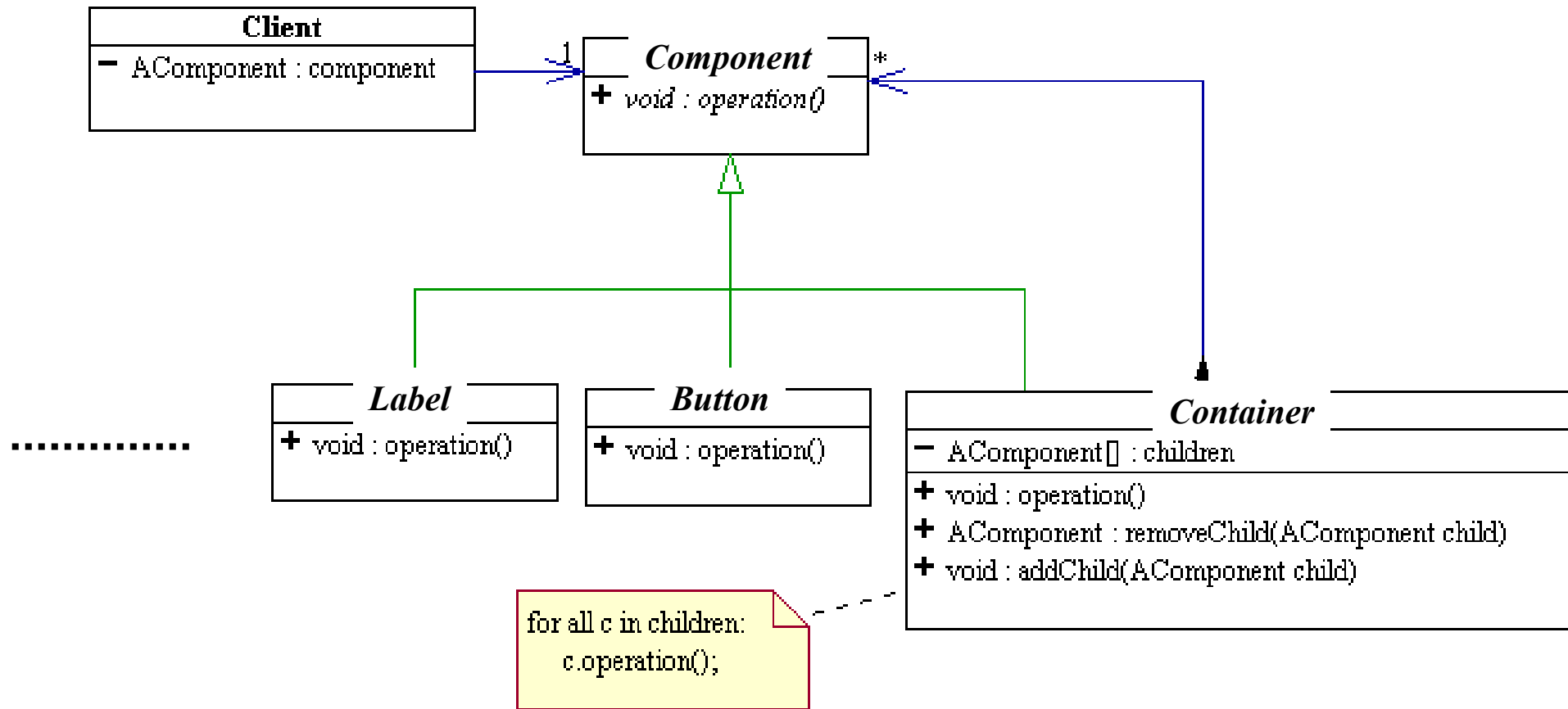
- **Modèle de conception réutilisables**
- **Assemblage de classes pour un discours plus clair**
- **Un modèle == plusieurs classes == Un nom de Pattern**
- **Les librairies standard utilisent ces Patterns**
 - L'API AWT utilise le modèle composite ???
 - Les évènements de Java sont dérivés du Pattern Observateur ???
 - Etc...
- **Une application = un assemblage de plusieurs patterns**

la bibliothèque graphique utilise un composite ?

- Le pattern Composite, recherche sur le web



la bibliothèque graphique utilise un composite :



À la place de

The screenshot shows a Mozilla Firefox browser window titled "Component (Java 2 Platform SE 5.0) - Mozilla Firefox". The address bar shows the file path: `file:///c:/Program%20Files/Java/jdk-1_5_0-doc/docs/a`. The browser has several tabs open, including "ESIEE.BureauD.in4...", "CNAM: SquirrelMail ...", "Sommaire des TP java", "designpatterns.pa...", and the active tab "Component (Jav...".

The main content area displays the Java™ 2 Platform Standard Ed. 5.0 documentation for the `Class Component` in the `java.awt` package. The navigation tabs include "Overview", "Package", "Class" (selected), "Use Tree", "Deprecated", "Index", and "Help".

The page content includes:

- Navigation links: [PREV CLASS](#), [NEXT CLASS](#), [FRAMES](#), [NO FRAMES](#)
- SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)
- Package: `java.awt`
- Class: `Class Component`
- Parent class: `java.lang.Object`
- Subclass: `java.awt.Component`
- All Implemented Interfaces: [ImageObserver](#), [MenuContainer](#), [Serializable](#)
- Direct Known Subclasses: [Button](#), [Canvas](#), [Checkbox](#), [Choice](#), [Container](#), [Label](#), [List](#), [Scrollbar](#), [TextComponent](#)

The bottom of the browser window features a search bar with the text "Rechercher :", navigation buttons for "Occurrence suivante" and "Occurrence précédente", a "Surligner" button, and a "Respecter la casse" checkbox. The status bar shows "Terminé" and "none".

Concepts de l'orienté objet

- **Le vocable Objet :**
- **Un historique ...**
- **Classe et objet (instance d'une classe)**
- **Etat d'un objet et données d'instance**
- **Comportement d'un objet et méthodes**
liaison dynamique
- **Héritage**
- **Polymorphisme**

Un historique

- **Algorithm + Data Structures = Program**
- **A + d = P** langage de type pascal
- **A + D = P** langage modulaire, Ada, modula-2
- **a + D = P** langage Orienté Objet

A + D = P, un exemple

- **surface** (triangle t) =
- **surface** (carré c) =
- **surface** (polygone_régulier p) =
-
- **perimetre** (triangle t) =
- **perimetre** (carré c) =
- **perimetre** (polygone_régulier p) =
-

- *usage : import de la **librairie** de calcul puis
carré unCarré; // une variable de type carré*
- **y = surface (unCarré)**

A + D = P

- type carré = structure
- longueurDuCote
- fin_structure;
- >>>-----<<<

- surface (carré c) =
- perimetre (carré c) =
- (carré c) =

- *usage : import du **module carré** puis*

carré unCarré; // une variable de type carré
- **y = surface (unCarré)**

$$A + D = P$$

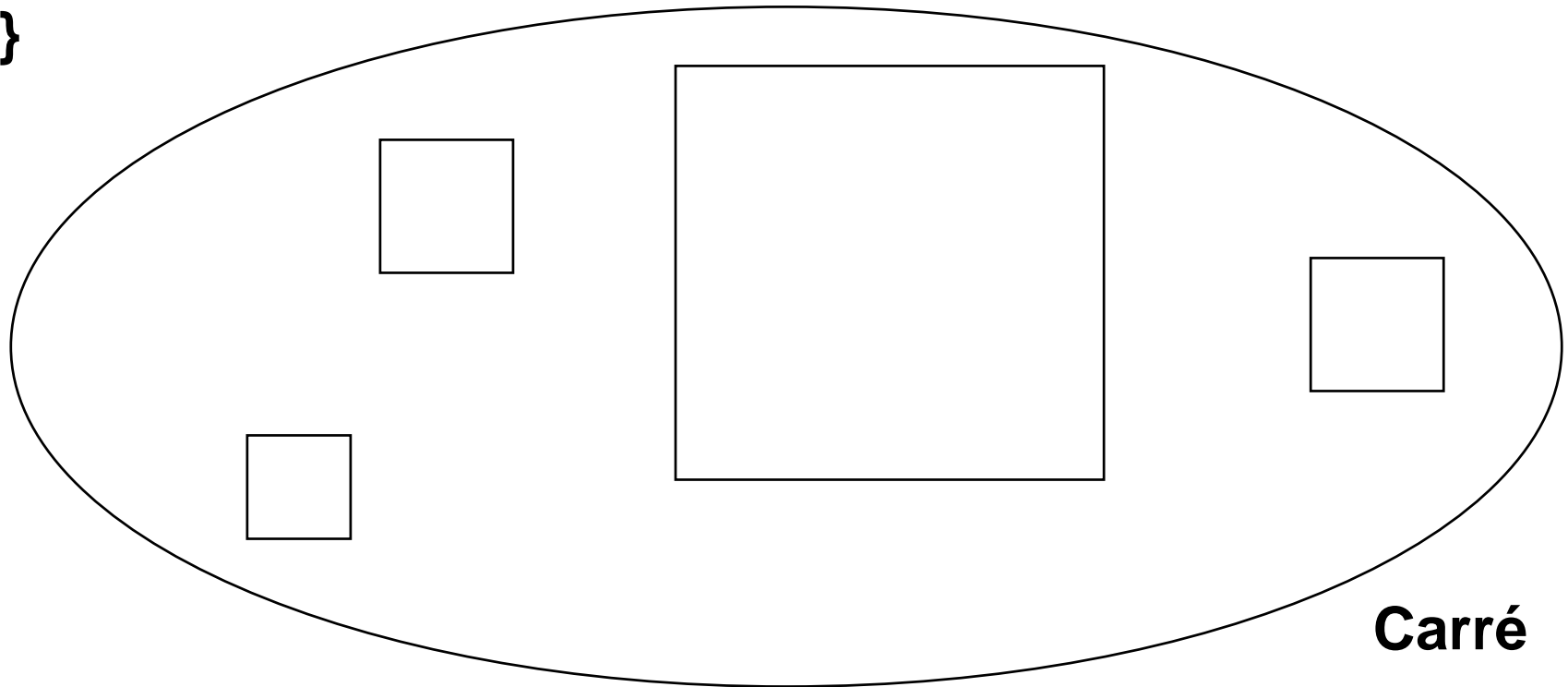
- classe **Carré** =
- longueurDuCote ...
- surface () =
- perimetre () =
- () =
- fin_classe;
- *usage : import de la **classe carré** puis*

carré unCarré; // une instance de la classe Carré
- **y = unCarré.surface ()**

Classe et objet (instance d'une classe)

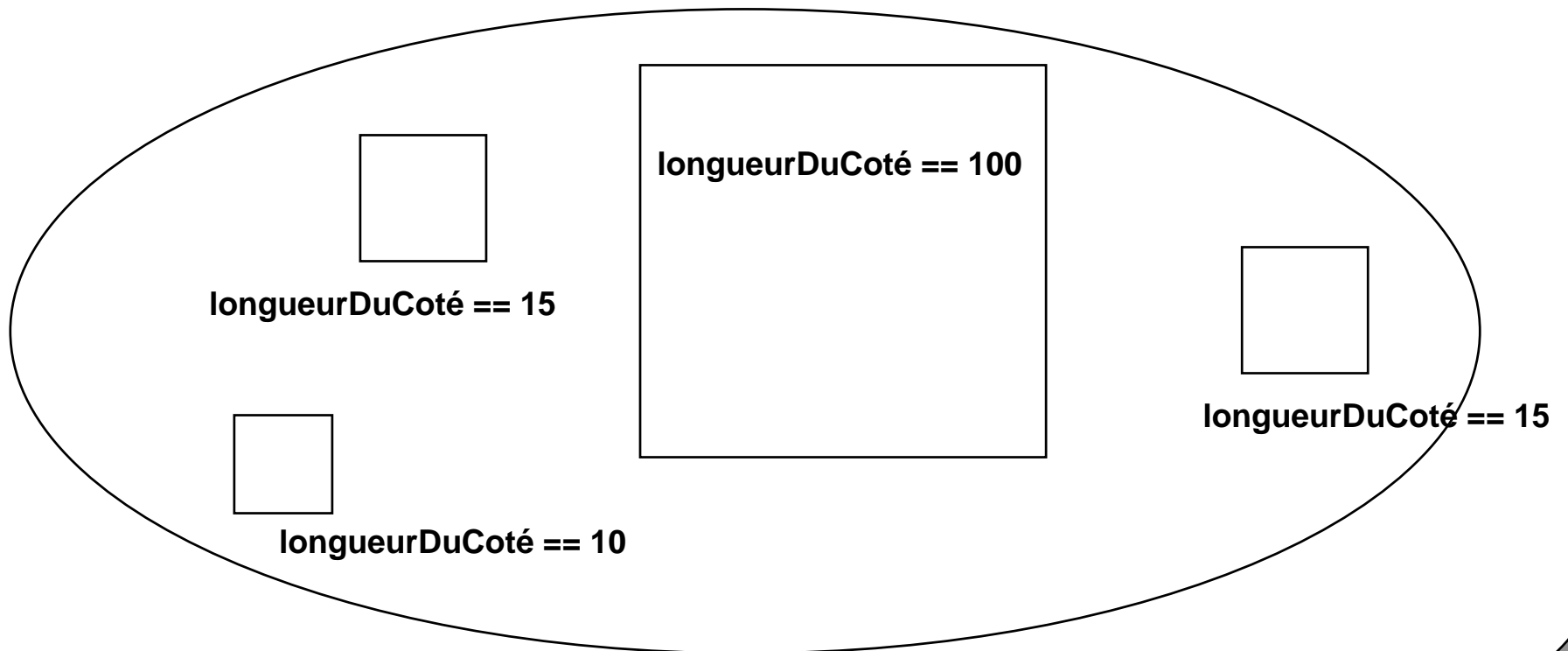
```
class Carré{
```

```
}
```



Etat d'un objet et données d'instance

```
class Carré{  
    int longueurDuCoté;  
}
```

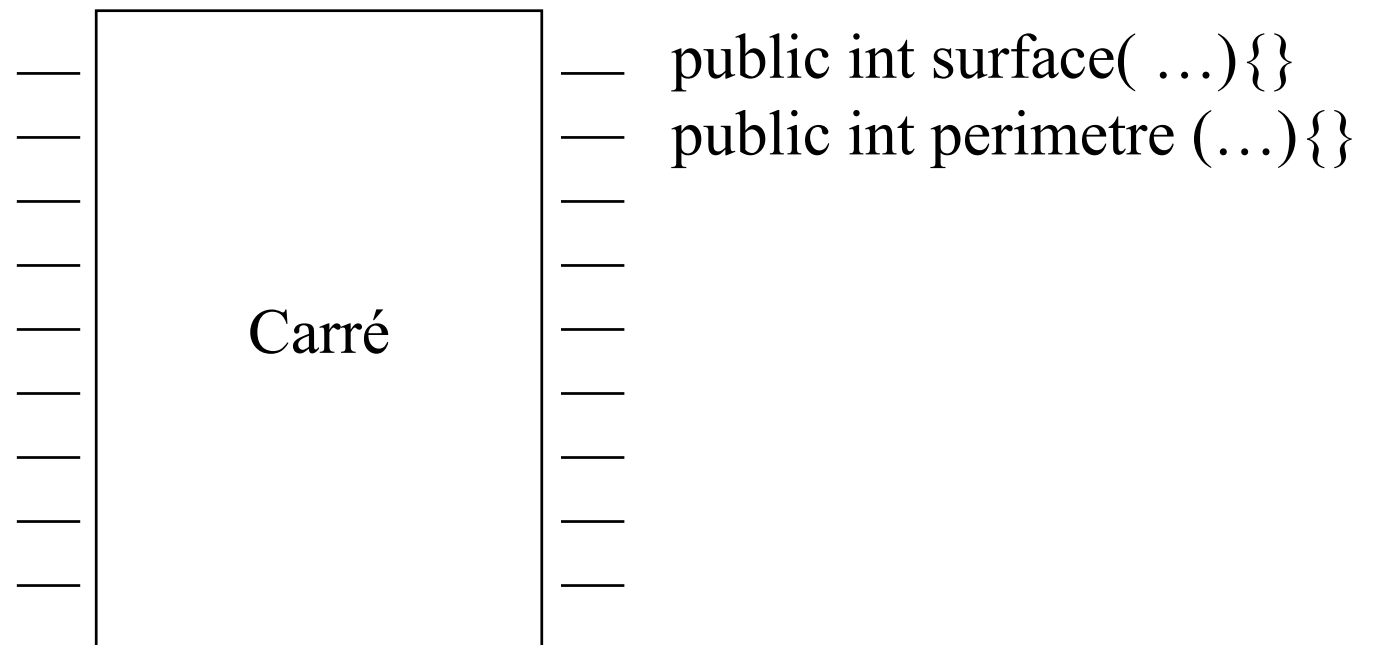


Classe et Encapsulation

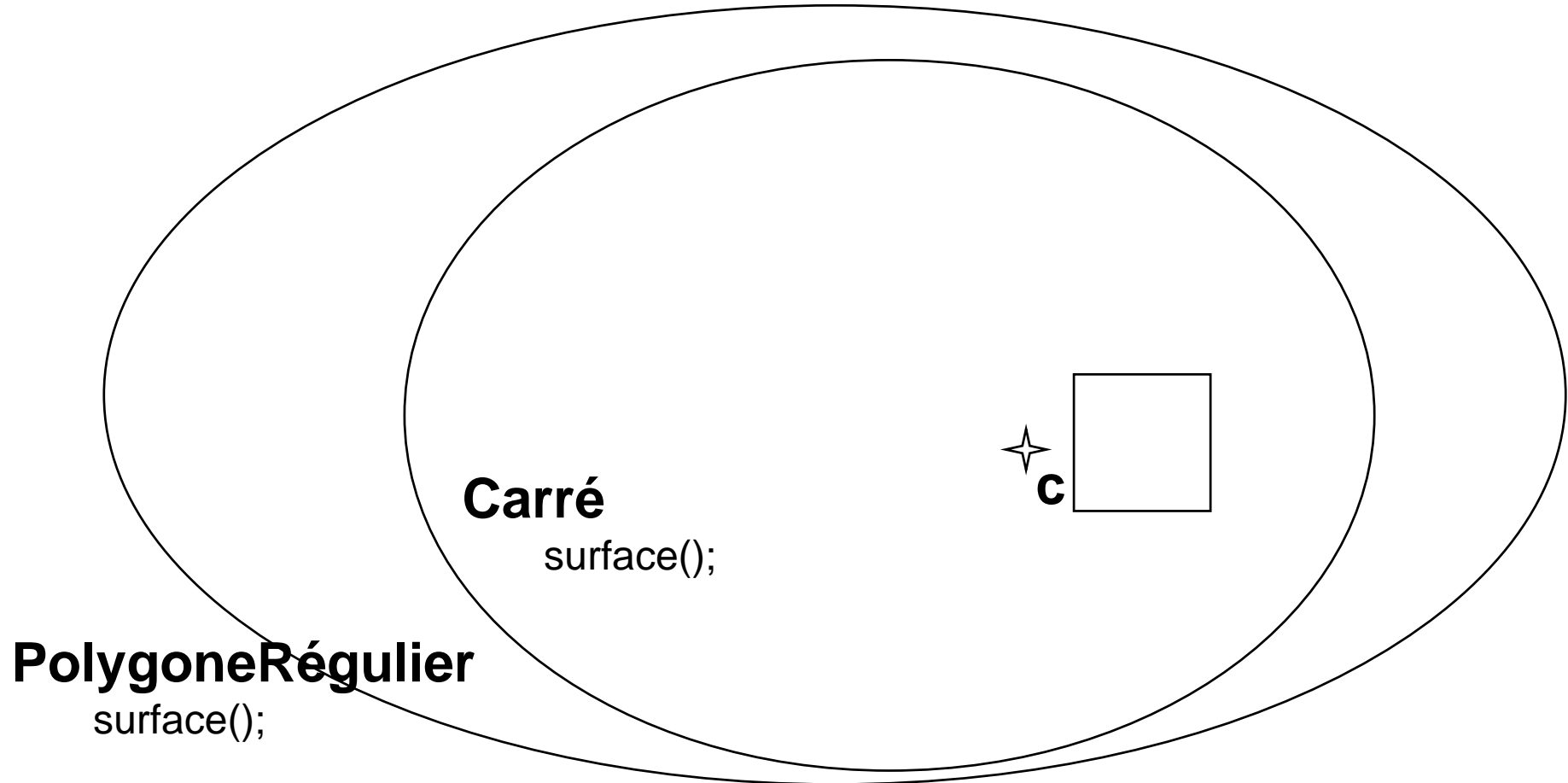
- **contrat avec le client**

interface publique

implémentation privée, ce sont des choix d'implémenteurs



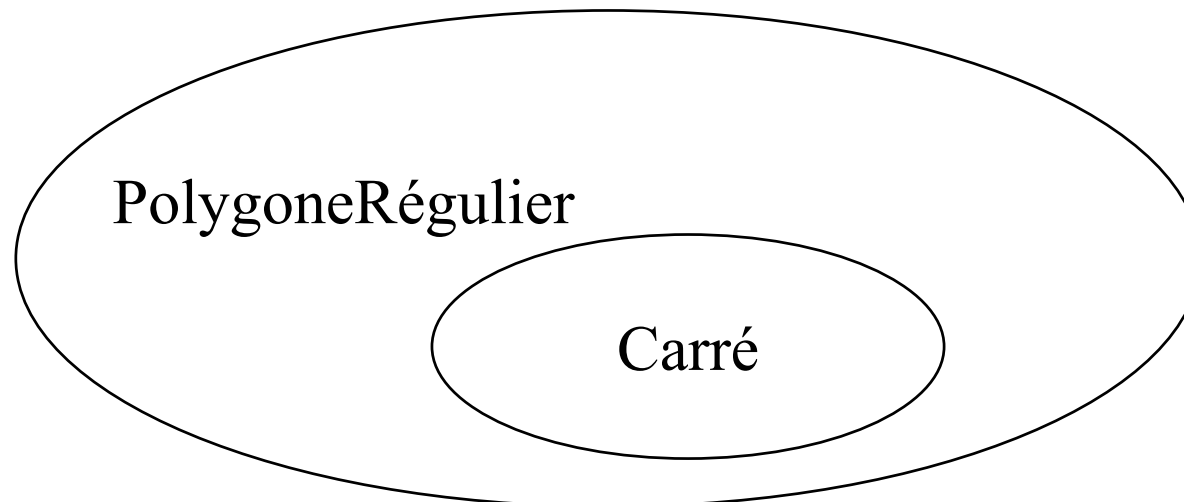
Comportement d'un objet et méthodes



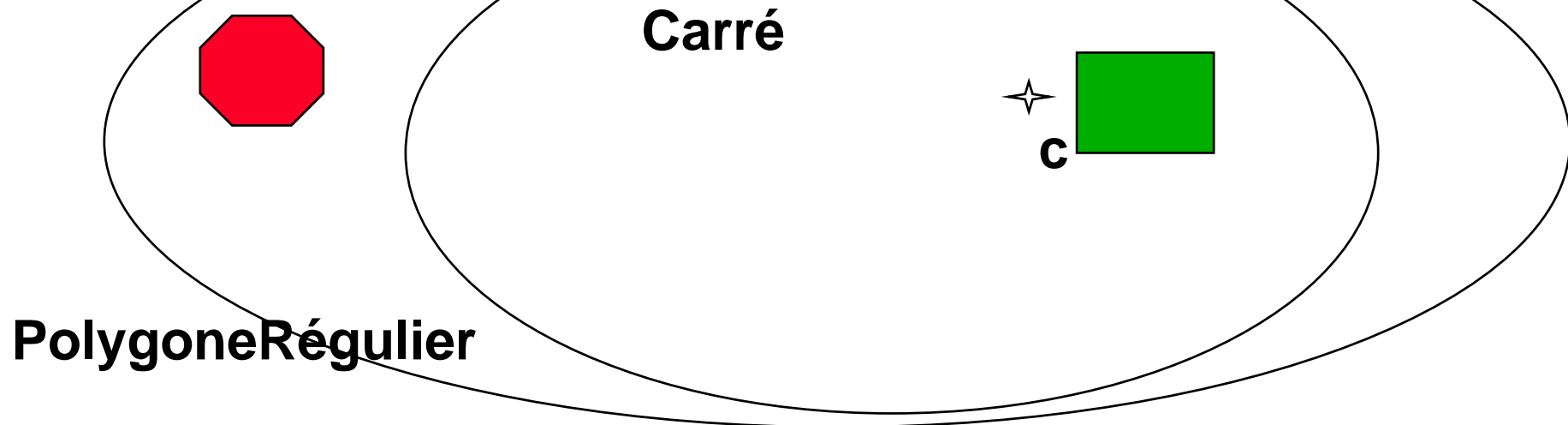
- **PolygoneRégulier p = c;**
- **p.surface(); ?**

Héritage et classification

- **définir une nouvelle classe en ajoutant de nouvelles fonctionnalités à une classe existante**
 - ajout de nouvelles fonctions
 - ajout de nouvelles données
 - redéfinition de certaines propriétés héritées (masquage)
- **Une approche de la classification en langage naturel**
- Les carrés **sont** des polygones réguliers (ce serait l'idéal...)



Les carrés sont des polygones, attention



- Les carrés sont de couleur verte
- Les polygones réguliers sont rouge
- ? Couleur d'un PolygoneRegulier de 4 côtés ?

Polymorphisme : définitions

- **Polymorphisme ad'hoc**

Surcharge(overloading),

plusieurs implémentations d'une méthode en fonction des types de paramètres souhaités, le choix de la méthode est résolu statiquement dès la compilation

- **Polymorphisme d'inclusion**

Redéfinition (overriding),

est fondé sur la relation d'ordre partiel entre les types, relation induite par l'héritage. si le type B est inférieur selon cette relation au type A alors on peut passer un objet de type B à une méthode qui attend un paramètre de type A, le choix de la méthode est résolu dynamiquement en fonction du type de l'objet receveur

- **Polymorphisme paramétrique ou généricité,**

consiste à définir un modèle de procédure, ensuite incarné ou instancié avec différents types, ce choix est résolu statiquement

extrait de M Baudouin-Lafon. La Programmation Orientée Objet. ed. Armand Colin

Polymorphisme ad'hoc

- `3 + 2` `3.0 + 2.5` `"bon" + "jour"`

- `out.print(3);` `out.print(3.0);` `out.print("bonsoir");`

le choix de la méthode est résolu statiquement dès la compilation

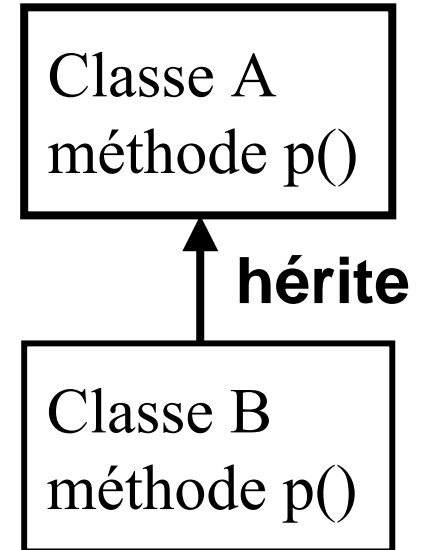
Polymorphisme d'inclusion

- **A a = new A(); a.p();**
- **B b = new B(); b.p();**
- **A a = new B(); a.p();**

- **void m(A a){
 a.p();
}**

m(new B()); m new(A());

le choix de la méthode est résolu dynamiquement en fonction du type de l'objet receveur



Polymorphisme paramétrique

- Une liste homogène

- **Class** `Liste<T>`{

`void add(T t) ...`

`void remove(T t) ...`

`...`

`}`

`List<Integer> li = new List<Integer>(); li.add(new Integer(4));`

`List<A> la = new List<A>(); la.add(new A()); la.add(new B());`

incarné ou instancié avec différents types, ce choix est résolu
statiquement

Le premier exemple en Java : la classe Carre

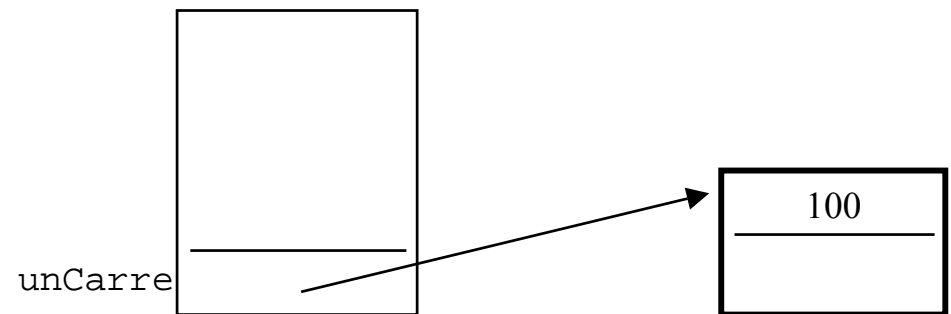
```
public class Carre {
    private int longueurDuCote;

    public void initialiser(int longueur){
        longueurDuCote = longueur;
    }

    public int surface(){
        return longueurDuCote * longueurDuCote;
    }

    public int perimetre(){
        return 4*longueurDuCote;
    }
}

// un usage de cette classe
Carre unCarre = new Carre();
unCarre.initialiser(100);
int y = unCarre.surface()
```



Java : introduction

- **Une approche traditionnelle d'un langage impératif**
- **types primitifs**
- **variables de type primitif**
- **types structurés**
- **Opérateurs**
- **Instructions**

Types primitifs

- **entier**

signés seulement

type **byte** (8 bits), **short** (16 bits), **int** (32 bits), **long** (64 bits)

- **flottant**

standard IEEE

type **float**(32 bits), **double** (64bits)

- **booléen**

type **boolean** (true,false)

- **caractère**

unicode,

type **char** (16 bits) <http://www.unicode.org>

Variables de type primitif

- *type nom_de_la_variable;*
- *type nom1,nom2,nom3;*
- *type nom_de_la_variable = valeur;*

- **exemples :**
 - `int i;`
 - `int j = 0x55AA0000;`
 - `boolean succes = true;`

- **l'adresse d'une variable ne peut être déterminée**

- **le passage de paramètre est par valeur uniquement**

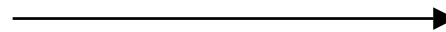
Passage de paramètres par valeur uniquement

```
static int plusGrand(int x, int y, int z){  
    ...  
}
```

```
int a=2, b=3, c=4;  
plusGrand(a,b,4);
```

c	4
b	3
a	2

Appel de plusGrand(a,b,4);



z	4
y	3
x	2
c	4
b	3
a	2

Type entier et Changement de type

- **Automatique**

si la taille du type destinataire est supérieure

```
byte a,b,c;
```

```
int d = a+b/c;
```

- **Explicite**

```
byte b = (byte)500;
```

```
b = (byte)( b * 2);           // b * 2 promue en int
```

*par défaut la constante numérique est de type int,
suffixe L pour obtenir une constante de type long 40L*

*par défaut la constante flottante est de type double,
suffixe F pour obtenir une constante de type float 40.0F*

Conversions implicites

- **Automatique**

si la taille du type destinataire est supérieure

byte -> short,int,long,float,double

short -> int, long, float, double

char -> int, long, float, double

int -> long, float, double

long -> float, double

float -> double

Application Java, un exemple

```
public class Application{
```

Nom de la classe -->
fichier Application.java

```
public static void main( String args[]){
```

Point d'entrée unique
la procédure "main"
args : les paramètres de
la ligne de commande

```
int i = 5;
```

variable locale à "main"

```
i = i * 2;
```

```
System.out.print(" i est égal à ");  
System.out.println( i);
```

Instructions

affichage

```
}
```

```
}
```

La classe Conversions : Conversions.java

```
public class Conversions{

    public static void main( String args[]){
        byte b;short s;char c;int i;long l;float f;double d;

        b=(byte) 0; s = b; i = b; l = b; f = b; d = b;

        i = s; l = s; f = s; d = s;

        i = c; l = c; f = c; d = c;

        l = i; f = i; d = i;

        f = l; d = l;

        d = f;
    }
}
```

Type caractère

- **Java utilise le codage Unicode**
- **représenté par 16 bits**
- **\u0020 à \u007E code ASCII, Latin-1**
 - \u00AE ©**
 - \u00BD / la barre de fraction ...**
- **\u0000 à \u1FFF zone alphabets**
 -**
 - \u0370 à \u03FF alphabet grec**
 -**
- **<http://www.unicode.org>**

Opérateurs

- **Arithmétiques**

`+, -, *, /, %, ++ +=, -=, /=, %=, --,`

Syntaxe C

- **Binaires**

`~, &, |, ^, &=, |=, ^=`

`>>, >>>, <<, >>=, >>>=, <<=`

- **Relationnels**

`==, !=, >, <, >=, <=`

Syntaxe C

- **Booléens**

`&, |, ^, &=, |=, ^=, ==, !=, !, ?:`


`||, &&`

Opérateurs booléens et court-circuits, exemple

```
• 1 public class Div0{
• 2     public static void main( String args[]){
• 3         int den = 0, num = 1;
• 4         boolean b;
• 5
• 6         System.out.println("den == " + den);
• 7
• 8         b = (den != 0 && num / den > 10);
• 9
• 10        b = (den != 0 & num / den > 10);
• 11    }
• 12}
```

***Exception in thread "main" java.lang.ArithmeticException :
/ by zero at Div0.main(Div0.java:10)***

Précédence des opérateurs



+	()	[]	.	
++	--	~	!	
*	/	%		
+	-			
>>	>>>	<<		
>	>=	<	<=	
==	!=			
&				
^				
&&				
?:				
=	op=			

- `int a = 1, b = 1, c=2;`
- `int x = a | 4 + c >> b & 7 | b >> a % 3; // ??? Que vaut x ???`

Type structuré : tableau

- **Déclarations de tableaux**

```
int[] mois; // mois est affecté à null ....  
ou int[] mois = new int[12];  
ou int[] mois={31,28,31,30,31,30,31,31,30,31,30,31};
```

- **Déclarations de tableaux à plusieurs dimensions**

```
double [][] m= new double [4][4];
```

- **Accès aux éléments**

le premier élément est indexé en 0
vérification à l'exécution des bornes, levée d'exception

- **En paramètre**

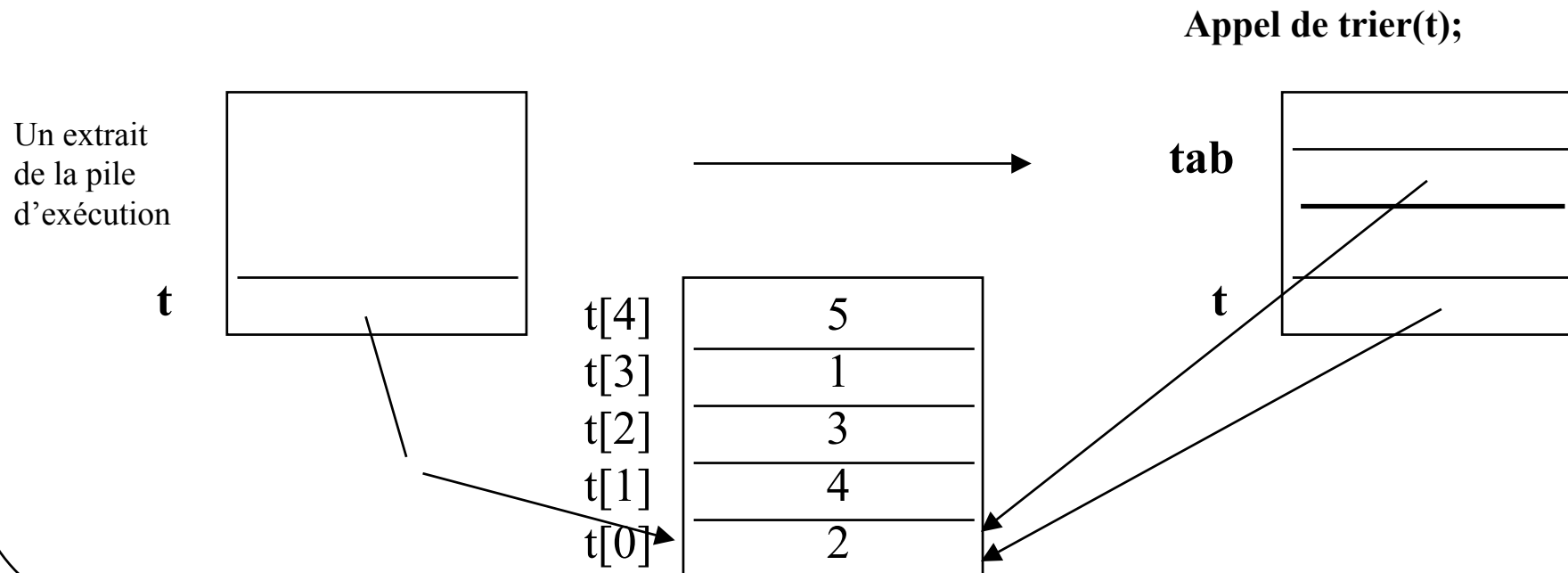
la variable de type tableau est une référence,
le passage par valeur de Java ne peut que transmettre la référence
sur le tableau

2 syntaxes autorisées : int mois[] ou int[] mois; la seconde est préférée !,
la première est devenue ancestrale ...

Passage de paramètres par valeur uniquement

```
static void trier(int[] tab){  
    tab[0] = 8; // --> t[0] == 8;
```

```
int[] t = {2,4,3,1,5};  
trier(t);
```



Instructions de contrôle

- **branchement**

if else, break, switch, return

// syntaxe C

- **Itération**

while, do-while, for, ' ', continue

// syntaxe C

- **Exceptions**

try catch finally, throw

Instruction de branchement, *if else*

- ***if (expression-booleenne) instructions1; [else instructions2]***

```
public class IfElse{
    public static void main( String[] args){
        int    mois = 4;
        String saison;

        if ( mois == 12 || mois == 1 || mois == 2){
            saison = "hiver";
        } else if ( mois == 3 || mois == 4 || mois == 5){
            saison = "printemps";
        } else if ( mois == 6 || mois == 7 || mois == 8){
            saison = "ete";
        } else if ( mois == 9 || mois == 10 || mois == 11){
            saison = "automne";
        } else {
            saison = "";
        }
        System.out.println("Avril est au " + saison + ".");
    }
}
```

if (false) : compilation conditionnelle

```
public class IfElseDebug{
    public static final boolean DEBUG = false;

    public static void main( String[] args){
        int  mois = 4;

        if (DEBUG) System.out.println( " mois = " + mois);
        else mois++;
    }
}
```

DOS> javap -c IfElseDebug

```
Method void main (java.lang.String[])
0  iconst_4          // int mois = 4;
1  istore_1
2  iinc 1 1          // mois++;
5  return
```

Instructions de branchement, *switch case*

- ***switch (expression) {***
- ***case value1 :***
- ***break;***
- ***case value2 :***
- ***.....***
- ***case value3 :***
- ***break;***
- ***case valueN :***
- ***break;***
- ***default :***
- ***}***

swicth case, exemple

```
public class SwitchSaison{
    public static void main( String[] args){
        int  mois = 4;    String saison;

        switch (mois){
            case 12: case 1: case 2:
                saison = "hiver";
                break;
            case 3: case 4: case 5:
                saison = "printemps";
                break;
            case 6: case 7: case 8:
                saison = "ete"; break;
            case 9: case 10: case 11:
                saison = "automne"; break;
            default:
                saison = "";
        }
        System.out.println("Avril est au " + saison + ".");
    }
}
```


Itérations

- **while** (*expression*) {
- *instructions*
- }

- **for** (*initialisation; terminaison; itération*) *instructions;*
- \Leftrightarrow
- *initialisation;*
- **while** (*terminaison*){
- *instructions;*
- *itération;*
- }

Itération, *for(; ;),* exemple

```
public class Mois{
    public static void main( String[] args){

        String[] mois={"janvier","fevrier","mars","avril","mai","juin",
                       "juillet","aout","septembre","octobre","novembre","decembre"};
        int[] jours={31,28,31,30,31,30,31,32,30,31,30,31};
        String printemps = "printemps";
        String ete = "ete";
        String automne = "automne";
        String hiver = "hiver";
        String[] saisons={ hiver,hiver,printemps,printemps,printemps,ete,ete,ete,
                           automne,automne,automne,hiver};

        for(int m = 0; m < 12; m++){
            System.out.println(mois[m] + " est au/en " +saisons[m] + " avec " +
                               jours[m] + " jours.");
        }
    }
}
```

break, continue

- **break;** *fin du bloc en cours*
- **continue;** *poursuite immédiate de l'itération*

- for (.....){
- →
- | **continue;**
- ————

- **etiquette :** for(.....){
- for (.....){
-
- **continue etiquette;**
-
- }
- }

Les exceptions: présentation

- Condition **anormale** d'exécution d'un programme

```
try {  
    instructions;  
    instructions;  
    instructions;  
} catch( ExceptionType1 e){  
    traitement de ce cas anormal de type ExceptionType1;  
} catch( ExceptionType2 e){  
    traitement de ce cas anormal de type ExceptionType2;  
    throw e; // l'exception est propagée  
              // ( vers le bloc try/catch) englobant  
} finally (  
    traitement de fin de bloc try ;  
}
```

Les exceptions : exemple

Soit l'instruction suivante :

```
m = Integer.parseInt(args[0]);
```

Au moins deux erreurs possibles :

1) args[0] n'existe pas

-> **ArrayIndexOutOfBoundsException**

2) le format de 'args[0]' n'est pas celui d'un nombre

-> **NumberFormatException**

Les exceptions : exemple

```
public class MoisException{
    public static void main( String[] args){
        String[] mois={"janvier","fevrier","mars","avril","mai","juin",
            "juillet","aout","septembre","octobre","novembre","decembre"};
        int[] jours={31,28,31,30,31,30,31,32,30,31,30,31};
        String printemps = "printemps"; String ete = "ete";
        String automne = "automne"; String hiver = "hiver";
        String[] saisons={hiver,hiver,printemps,printemps,printemps,ete,ete,ete, automne,automne,automne,hiver};

        int m;
        try{
            m = Integer.parseInt(args[0]) -1;
            System.out.println(mois[m] + " est au/en " +saisons[m] + " avec " + jours[m] + " j.");
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("usage : DOS>java MoisException unEntier[1..12]");
        }catch(NumberFormatException e){
            System.out.println("exception " + e + " levee");
        }finally{
            System.out.println("fin du bloc try ");
        }
    }
}
```

L 'exemple initial avec une division par zéro

```
1 public class Div0{
2     public static void main( String args[]){
3         int den = 0, num = 1;
4         boolean b;
5
6         System.out.println("den == " + den);
7
8         b = (den != 0 && num / den > 10);
9
10        b = (den != 0 & num / den > 10);
11    }
12}
```

*// Il existe un donc un bloc try/catch prédéfini
// interne à la machine virtuelle*

```
try{
    Div0.main( {" " });
}catch(RuntimeException e){
    System.err.println(e);
}
```

Les exceptions sont des classes

```
java.lang.Object
```

```
|
```

```
+--java.lang.Throwable
```

```
|
```

```
+--java.lang.Exception
```

```
|
```

```
+--java.lang.RuntimeException
```

```
|
```

```
+--java.lang.IndexOutOfBoundsException
```

```
|
```

```
+--java.lang.ArrayIndexOutOfBoundsException
```

```
+--java.lang.RuntimeException
```

```
ArithmeticException, ArrayStoreException, CannotRedoException,
```

```
CannotUndoException, ClassCastException, CMMException,
```

```
ConcurrentModificationException, EmptyStackException, IllegalArgumentException,
```

```
IllegalMonitorStateException, IllegalPathStateException, IllegalStateException,
```

```
ImagingOpException, IndexOutOfBoundsException, MissingResourceException,
```

```
NegativeArraySizeException, NoSuchElementException, NullPointerException,
```

```
ProfileDataException, ProviderException, RasterFormatException, SecurityException,
```

```
SystemException, UnsupportedOperationException
```


if(DEBUG) + throw new RuntimeException

```
public class IfElseDebug{
    public static final boolean DEBUG = false;

    public static void main( String[] args){
        int  mois = 4;

        if (DEBUG) assert(mois==4);
        else mois++;
    }

    public static void assert(boolean b){
        if (!(b)) throw new RuntimeException(" assertion fausse  " ) ;
    }
}
DOS> javap -c IfElseDebug
Method void main (java.lang.String[])
0  iconst_1
1  istore_1
2  iinc 1 1
5  return
```

NullPointerException

```
static void trier(int[] tab){  
    // levée d'une exception instance de la classe  
    // NullPointerException,  
}
```

```
int[] t; /** t == null; **/  
trier(t); // filtrée par la machine
```

ou bien

```
    int[] t;  
    try{  
        trier(t);  
    }catch(NullPointerException e){}
```

Classe : Syntaxe, champs et méthodes "statiques"

- **public class** NomDeClasse{
 static type variable1DeClasse;
 static type variable2DeClasse;
 public static type variableNDeClasse;

 static type nom1DeMethodeDeClasse(listeDeParametres) {

 }
 static type nom2DeMethodeDeClasse(listeDeParametres) {

 }
 static type nomNDeMethodeDeClasse(listeDeParametres) {

 }
 public static void main(String args []){ } //
• }

Notion Variables et méthodes globales

Accès aux champs de classe

- Le mot clé **static**
- **accès en préfixant par le nom de la classe**

exemple la classe prédéfinie "Integer"

```
package java.lang;  
public class Integer ... {  
    public static final int MAX_VALUE= ...;  
    ...  
    public static int parseInt(String s){ ...}  
    ...  
}
```

// accès :

```
int m = Integer.parseInt("33");
```

- **Opérateur "."**
appels de méthodes de classe (si elles sont accessibles)
accès aux champs de classe (si accessibles)

Variables et méthodes de classe

Variables de classes, (notion de variables globales)

```
public class MoisBis{
    static String[] mois={"janvier","fevrier","mars","avril","mai","juin",
        "juillet","aout","septembre","octobre","novembre","decembre"};
    static int[] jours={31,28,31,30,31,30,31,32,30,31,30,31};
    static String printemps = "printemps";
    static String ete = "ete";
    static String automne = "automne";
    private static String hiver = "hiver";
    public static String[]
        saisons={hiver,hiver,printemps,printemps,printemps,ete,ete,ete,
            automne,automne,automne,hiver};
    public static void main( String[] args){
        for(int m = 0; m < 12; m++){
            System.out.println(mois[m] + " est au/en " +saisons[m] + " avec " +
                jours[m] + " jours.");
        }
    }
}
```

Définition d'une Classe

- **modificateurs d'accès aux variables et méthodes**

accès : public, private, protected et « sans »

final

static

- **Déclararation et implémentation**

dans le même fichier ".java" (NomDeLaClasse.java)

documentation par l'outil *javadoc* et `/** */`

une seule classe publique par fichier ".java"

- **class Object**

toutes les classes héritent de la classe "Object"

méthodes de la classe Object

String toString();

String getClass();

Object clone();

boolean equals(Object);

void finalize()

Le bloc static

exécuté une seule fois au chargement de la classe

```
public class MoisTer{
    static String[] mois ={"janvier", "fevrier", "mars", "avril", "mai", "juin",
                          "juillet", "aout", "septembre", "octobre", "novembre", "decembre"};
    static int[] jours ={31,28,31,30,31,30,31,32,30,31,30,31};
    static String printemps,ete,automne,hiver;
    static String[] saisons ={ hiver,hiver,printemps,printemps,printemps,ete,ete,ete,
                              automne,automne,automne,hiver};

    public static void main( String[] args){
        for(int m = 0; m < 12; m++){
            System.out.println(mois[m] + " est au/en " +saisons[m] + " avec " +
                               jours[m] + " jours.");
        }
    }

    static{
        printemps = "printemps"; ete = "ete";
        automne = "automne"; hiver = "hiver";
    }
}
```

Application Java

- **Caractéristiques**

 - autonome, sans navigateur

 - une ou plusieurs classes

 - les classes sont chargées dynamiquement

- **Le point d'entrée *main***

 - l'application doit comporter une classe avec la méthode *main*

 - sa signature est `public static void main(String[] args)`

 - appelée par la commande `DOS> java` suivie du nom de la classe ayant implémentée la méthode *main*

- **Passage d'argument(s)**

 - sur la ligne de commande

 - `DOS> java NomDeClasse Octobre 30`

- **Gestion des paramètres dans l'application**

 - tous ces paramètres sont de "type" `String`

 - conversion en entier par exemple

 - `int x = Integer.parseInt(args[0]); // classe Integer`

Application Java : style d'écriture(1)

```
public class ApplicationJavaStyle1{

    public static void main(String[] args){
        // variables locales
        int x, y;

        // appels de méthodes statiques (uniquement)
        proc(x,y);
    }

    public static void proc( int i, int j){
        ...
    }

}
```

Application Java : style d'écriture(2)

```
public class ApplicationJavaStyle2{
    // Variables statiques
    static int x;
    static int y;

    public static void main(String[] args){

        //usage de variables statiques (uniquement)
        // appels de méthodes statiques (uniquement)
        proc(x,y);
    }

    public static void proc( int i, int j){
        ...
    }
}
```

Package

- **Fonction**

Unité logique par famille de classes

découpage hiérarchique des paquetages

(ils doivent être importés explicitement sauf java.lang)

- **Buts**

espace de noms

restriction visibilité

- **Instructions**

package pkg1[.pkg2[.pkg3];

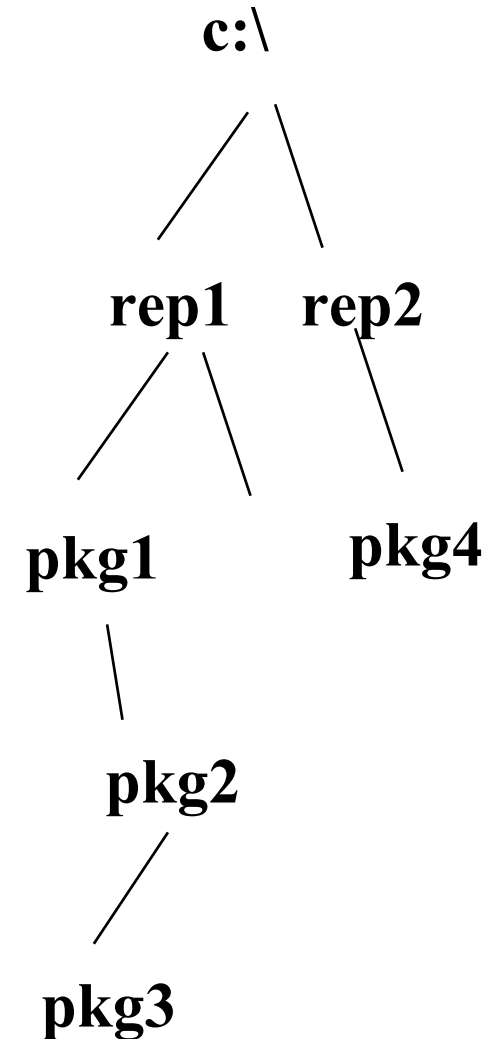
les noms sont en minuscules

c'est la première instruction du source java

import pkg1[.pkg2[.pkg3].(nomdeclasse/);*

liés aux options de la commande de compilation

dos> javac -classpath .;c:\rep1;c:\rep2



Paquetages prédéfinis

- **le paquetage `java.lang.*` est importé implicitement**

ce sont les interfaces : *Cloneable*, *Comparable*, *Runnable*

et les classes : `Boolean`, `Byte`, `Character`, `Class`, `ClassLoader`, `Compiler`,
`Double`, `Float`, `InheritableThreadLocal`, `Long`, `Math`, `Number`,
`Object`, `Package`, `Process`, `Runtime`, `RuntimePermission`,
`SecurityManager`, `Short`, `StrictMath`, `String`, `StringBuffer`,
`System`, `Thread`, `ThreadGroup`, `ThreadLocal`,
`Throwable`, `Void`,

toutes les classes dérivées de `Exception`, `ArithmeticException`,.....

et celles de `Error`, `AbstractMethodError`,.....

- **`java.awt.*` `java.io.*` `java.util.*` **

Exercice : <http://java.cnam.fr/public~1/iagl99/douin.htm> --> TP1

1) Développer une application Java effectuant la conversion d'une valeur exprimée en degré fahrenheit en degré celcius,

$$^{\circ}\text{C} = 5/9 * (^{\circ}\text{F} - 32)$$

La documentation générée par l'utilitaire ' javadoc ' est demandée

<http://www.javasoft.com/products/jdk/javadoc/writingdoccomments.html>

Les valeurs en $^{\circ}\text{F}$ à convertir doivent être sur la ligne de commande,

(voir `java.lang.Float`, `java.lang.Integer`, <http://java.sun.com/docs/books/jls/html/index.html>)

compilation, DOS> `javac Fahr.java`

documentation, DOS> `javadoc Fahr.java`

execution, DOS> `java Fahr 20 40 60`

20 $^{\circ}\text{F}$ -> -6.6667 $^{\circ}\text{C}$

40 $^{\circ}\text{F}$ -> 4.4444 $^{\circ}\text{C}$

60 $^{\circ}\text{F}$ -> 15.5554 $^{\circ}\text{C}$

2) Ajouter à votre programme la gestion des exceptions susceptibles d'être levées (`NumberFormatException`, `ArrayIndexOutOfBoundsException`).
Produire une nouvelle documentation

Résumé

- **Types primitifs et type « Object » (et ses dérivés)**
- **Instructions analogue au langage C**
- **La classe : seule unité de compilation**
- **Variables de classe**
- **Application Java, methode de classe main**
- **Regroupement de classes en paquetage**