UNIVERSITÉ CÔTE D'AZUR

Diversité des Langages Informatiques et Styles de Programmation

Frédéric Mallet

Frederic.Mallet@univ-cotedazur.fr

UNIVERSITÉ CÔTE D'AZUR

Plan du cours

- Styles
  - Impératif/déclaratif/récursif/logique
  - Avec ou sans objets/classes
  - Données vs fonctions
- Polymorphisme
  - Unification
  - Abstraction

UNIVERSITÉ CÔTE D'AZUR

Algorithme

- Discret
  - Succession d'actions (impératif)
- Déterministe
  - Les mêmes entrées produisent les mêmes sorties
  - ≠ Stochastique
- Finitude
  - Terminaison après un nombre fini de pas
  - Semi-algorithme: ne termine pas nécessairement

UNIVERSITÉ CÔTE D'AZUR

La programmation impérative

- Programmation
  - concevoir un **algorithme** et réaliser un **programme** pour une exécution **automatique**
- Algorithme
  - Impératif: succession d'ordres
    - Pour calculer le maximum, je parcours les éléments un à un et je les compare à l'élément supposé maximum
  - Déclaratif
    - $\max(T) = \{ m \mid \forall t \in T, t \leq m \}$

UNIVERSITÉ CÔTE D'AZUR

Exemple de la factorielle

Impératif

Itératif

```
long factorielle(int n) {
  long res = 1;
  for(int i = 2; i<n; i++) {
    res = res * i;
  }
  return res;
}
```

Récursif

```
long factorielle(int n) {
  if(n==0) return 1;
  return n*factorielle(n-1);
}
```

Déclaratif

Fonctionnel / Applicatif

```
(factorielle (n) (fac n 1))
(fac (n acc)
  ((zero? n) acc)
  (fac (- n 1) (* n acc))
```

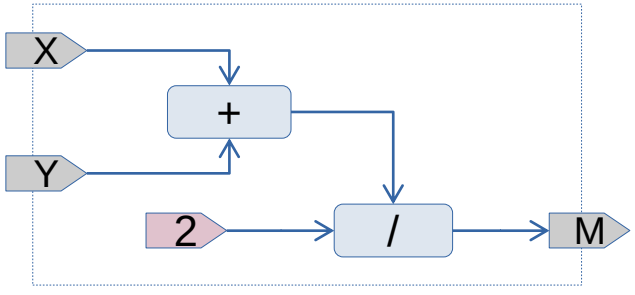
Logique

```
factoriel(0,1):-
  !.
factoriel(N,T):-
  N1 is N-1,
  factoriel(N1,T1),
  T is T1*N.
```

UNIVERSITÉ CÔTE D'AZUR

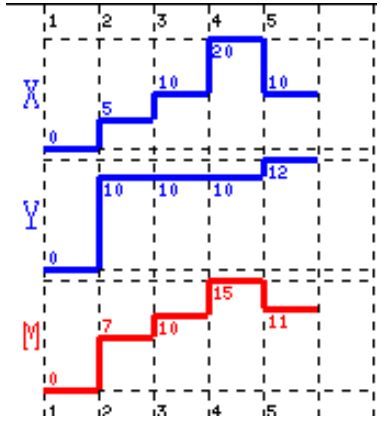
Langage flot de données synchrone

- Classique en automatique et conception de circuits
  - X, Y, S, M, 2 : sont des flots d'entiers infinis !




```
node Moyenne(X, Y : int) returns (M : int);
var S : int
let
   M = S / 2;
   S = X + Y;
tel
```

- Interprétation synchrone (temps logique : $\mathbb{N}$)
  - $\forall t \in \mathbb{N}, M_t = (X_t + Y_t) / 2$

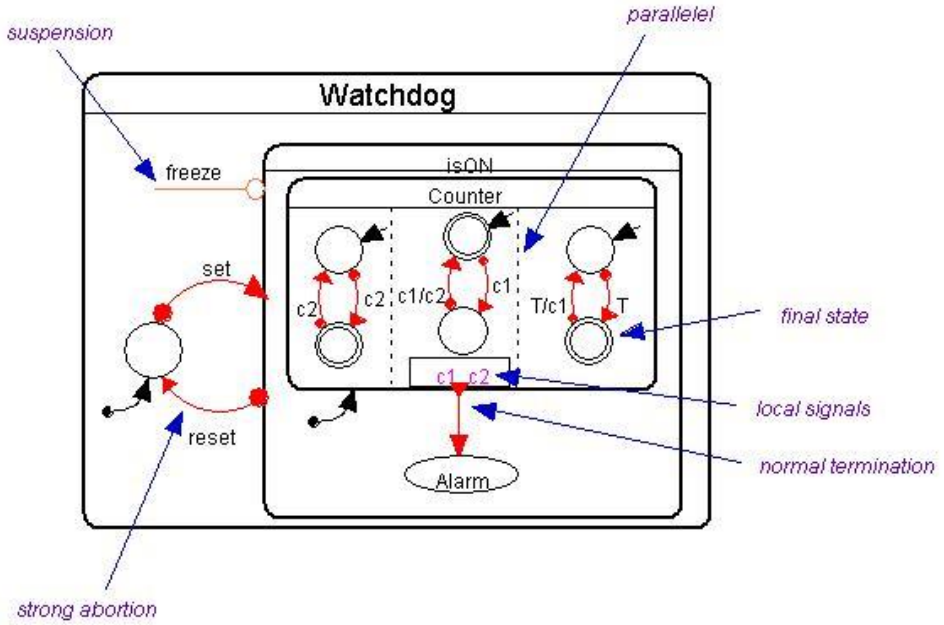



```
## STEP 1 #########
X (integer) ? 0
Y (integer) ? 5
M = 2
## STEP 2 #########
X (integer) ? 12
Y (integer) ? 18
M = 15
## STEP 3 #########
X (integer) ? 20
Y (integer) ? 10
M = 15
## STEP 4 #########
X (integer) ? 12
Y (integer) ? 15
M = 13
## STEP 5 #########
X (integer) ?
```

UNIVERSITÉ CÔTE D'AZUR

Langage impératif synchrone

```
module ABRO:
input A, B, R;
output O;

loop
  [ await A || await B ];
  emit O
each R

end module
```



UNIVERSITÉ CÔTE D'AZUR

Langage concurrent (temporisé)

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity signed_adder is
6   port
7   (
8     aclr : in   std_logic;
9     clk  : in   std_logic;
10    a    : in   std_logic_vector;
11    b    : in   std_logic_vector;
12    q    : out  std_logic_vector
13  );
14 end signed_adder;
15
16 architecture signed_adder_arch of signed_adder is
17   signal q_s : signed(a'high+1 downto 0); -- extra bit wide
18
19 begin -- architecture
20   assert(a'length >= b'length)
21     report "Port A must be the longer vector if different sizes!"
22     severity FAILURE;
23   q <= std_logic_vector(q_s);
24
25   adding_proc:
26   process (aclr, clk)
27     begin
28       if (aclr = '1') then
29         q_s <= (others => '0');
30       elsif rising_edge(clk) then
31         q_s <= ('0'&signed(a)) + ('0'&signed(b));
32       end if; -- clk'd
33     end process;
34
35 end signed_adder_arch;
```

UNIVERSITÉ CÔTE D'AZUR

La Programmation Orientée-Objet

- La **POO** guide la conception par
  - Un ensemble de concepts
    - abstraction, modularité, encapsulation, polymorphisme
  - Des langages et des outils qui supportent ces concepts
    - Classification vs. ~~prototype~~
    - Héritage ~~(multiple)~~
    - Typage : fort/~~faible~~, explicite/~~inféré~~

- Ses forces (supposées)
  - Reflète plus finement les objets du monde réel
    - Du code (plus) facile à maintenir
    - Plus stable : un changement s'applique à un sous-système facile à identifié et **isolé** du reste du système

Système de gestion d'un lycée

Complexité spatiale

Données/Objets

- Personne
  - Etudiant, enseignant, principal, secrétaire
- Diplôme
  - Année, matière, parcours
- Bulletin
  - Notes
  - Coefficients

Complexité temporelle

Fonctions

- Calculer la moyenne
- Calculer les taux d'encadrement
- Calculer le nombre de redoublants
- Calculer le taux de réussite au baccalauréat

Questions:

- Quelles sont propriétés d'un étudiant?
- Comment représenter une note ou une année?

UNIVERSITÉ CÔTE D'AZUR

Types de données

- Réalise un Type Abstrait de données (algorithmique)
- Le type **encode** les données
  - Domaine de valeurs (fini)
  - Opérations autorisées (arithmétiques, booléennes…)
- On distingue
  - Types **primitifs** (caractères, entiers, réels, booléens)
  - Types **composés**
    - Image = ensemble de pixels de couleur
    - Couleur = Rouge + Vert + Bleu

UNIVERSITÉ CÔTE D'AZUR

Nombres entiers relatifs ℕ

- Domaine de valeurs
  - 8 bits **byte** [-128,127]
  - 16 bits **short** [-32768, 32767]
  - 32 bits **int** $[-2^{31}, 2^{31}-1]$
  - 64 bits **long** $[-2^{63}, 2^{63}-1]$

`01`

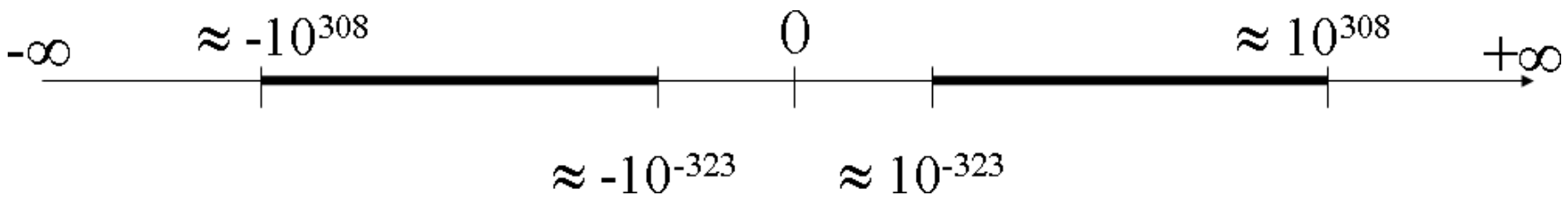
- Opérations autorisées
  - Affectation `=`
  - Comparaison `==, !=`
  - Opérations arithmétiques `+, -, *, /, %` (int, long)
  - Opérations binaires `&, |, ~, <<, >>` (int, long)

- Division entière: `23 = 3 * 7 + 2`
  - `/` quotient de la division entière `23/7` → 3
  - `%` reste de la division entière `23%7` → 2
  - `&` et binaire (différent du et logique) `23&7` → 7
  - 23 / 0 **Erreur !**

UNIVERSITÉ CÔTE D'AZUR

Nombres approchés $\mathbb{R}$

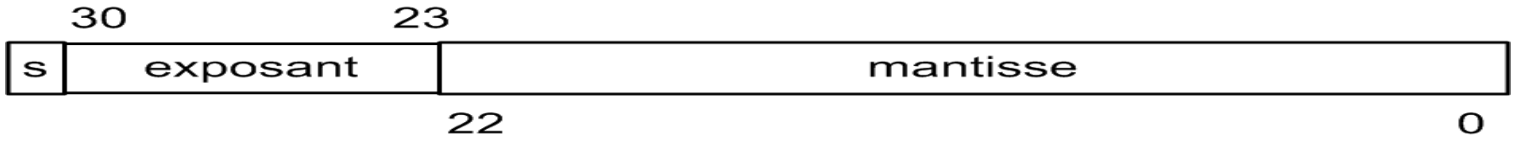
- Domaine de valeurs
  - 32 bits **float** ?? `12.5f`
  - 64 bits **double** ?? `-12.5, 1e-23`
- Opérations autorisées
  - Affectation `=`
  - Comparaison `==, !=`
  - Opérations arithmétiques `+, -, *, /`
  - Opérations binaires `&, |, ~, <<, >>`
- Particularités
  - Division approchée: 23.0 / 0 → +∞



UNIVERSITÉ CÔTE D'AZUR

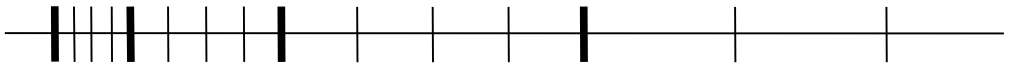
Nombres approchés – IEEE 754

- Codage sur 32 bits => 2^{32} valeurs
  - Complément à 2 (**int**) $[-2^{31}, 2^{31}-1]$
  - Virgule flottante (**float**) $[-2^{128}, 2^{128}]$



Signe	Exposant	Mantisse	Nombre représenté
S	E=]0,255[	M	$(-1)^S x 2^{E-127} x (1,M)_2$
S	0	>0	$\mathbf{(-1)}^S x 2^{-126} x (0,M)_2$
0	0	0	+0
1	0	0	-0
-	255	>0	NaN
0	255	0	+∞
1	255	0	–∞

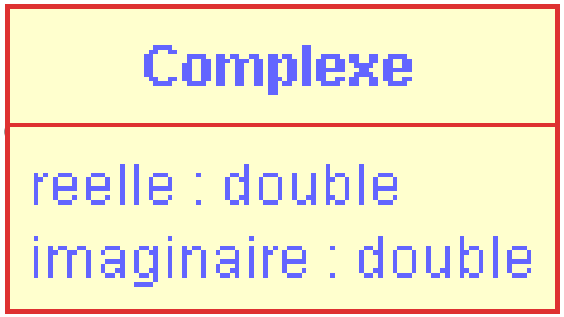
Not a Number !



UNIVERSITÉ CÔTE D'AZUR

Classe = Type composé

- Types primitifs normalisés par le langage
  - boolean, char, byte, short, int, long, float, double
- Composition => **champs/propriétés** (nom + type)
  - Couleur : rouge (8 bits) + vert (8 bits) + bleu (8 bits)
  - Image : ensemble de points de couleur
  - Nom : séquence/chaîne de caractères
  - Date : ?
- Compromis espace mémoire/temps de calcul
  - Couleur : RGB vs. YUV
  - Nombre complexe: c = re + i.im
    ou c = z.



UNIVERSITÉ CÔTE D'AZUR

Classe: encodage

- Couleur

```
class CouleurRGB {
  byte rouge;
  byte vert;
  byte bleu;
}

class CouleurYUV {
  byte luminance;
  byte chrominanceRouge;
  byte chrominanceBleue;
}
```

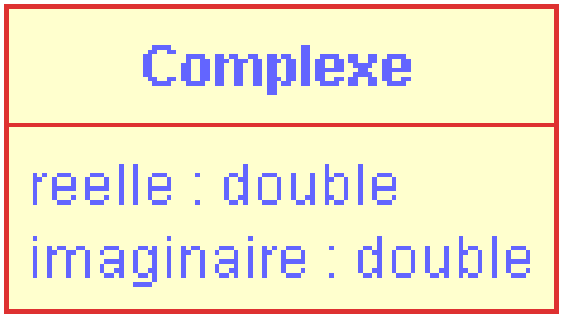
- Complexe

```
class ComplexeCartesien {
  double reelle;
  double imaginaire;
}

class ComplexePolaire {
  double module;
  double argument;
}
```

UNIVERSITÉ CÔTE D'AZUR

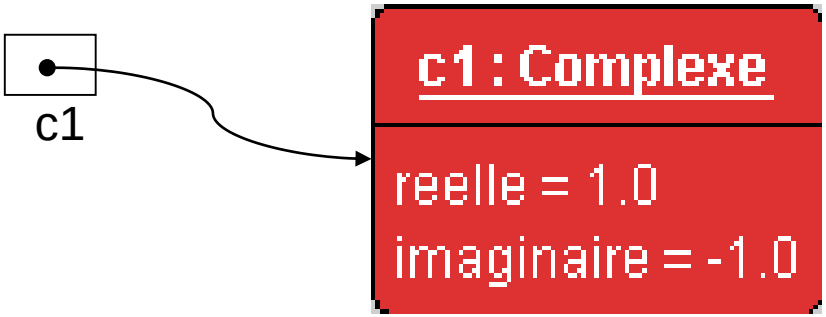
Objet = instance d'une classe

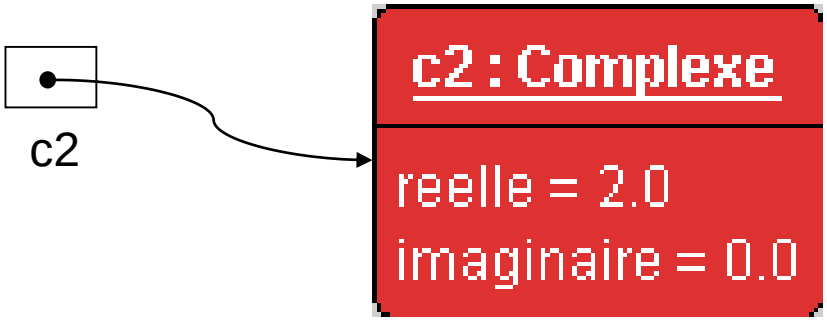
- Type → valeurs → variables
- **Classe → objets → références**




```
Complexe c1; // référence
c1 = new Complexe(1, -1);
```

```
Complexe c2; // référence
c2 = new Complexe(2, 0);
```





UNIVERSITÉ CÔTE D'AZUR

Méthode = opération sur les objets

- Norme d'un complexe

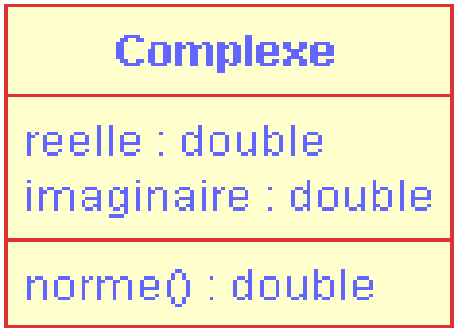
```
class ComplexeCartesien {
  double re;
  double im;

  double norme () {
     return re * re + im * im;
  }

  double argument () {
    return Math.atan2(im, re);
  }
}
```

```
class ComplexePolaire {
  double module;
  double argument;

  double norme() {
    return module * module;
  }

  double argument() {
    return argument;
  }
}
```

UNIVERSITÉ CÔTE D'AZUR

Notation pointée

- Pour accéder aux membres d'un objet
  - Membre = champs + méthodes



- Exemple
  - `Complexe c1 = new Complexe();`
  - `c1.reelle` désigne le champ `reelle` de `c1`
  - `c1.norme()` désigne la méthode `norme` de `c1`

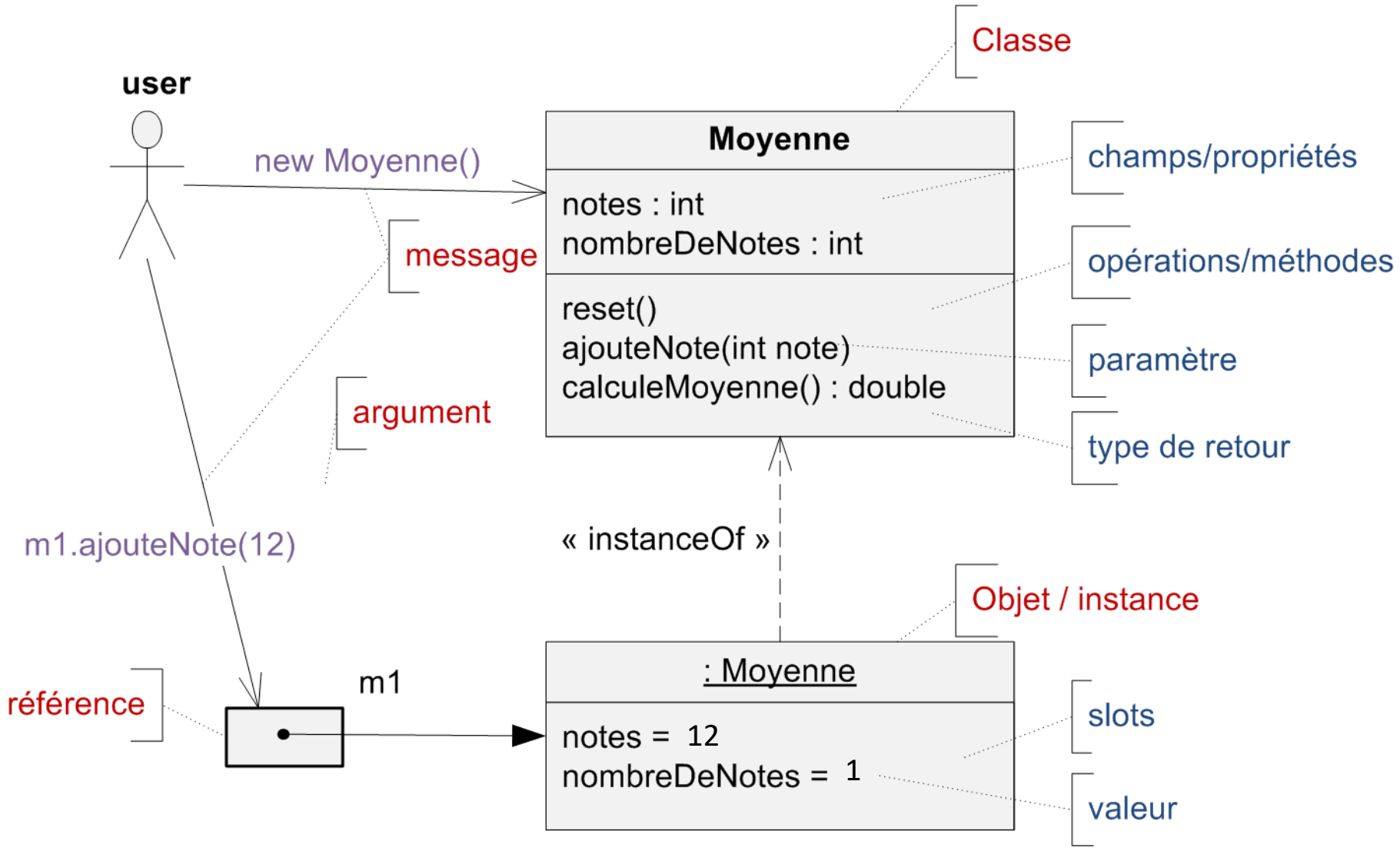
UNIVERSITÉ CÔTE D'AZUR

Classe Moyenne

```
class Moyenne { // bloc de déclaration de classe
  /** Somme des notes obtenues */
  int notes = 0 ;
  /** Nombre de notes obtenues */
  int nombreDeNotes = 0 ;

  /** Ajoute une note à la moyenne
   *  @param note nouvelle note obtenue */
  void ajouteNote (int note) {
     notes += note;
     nombreDeNotes += 1;
  }

  /** @return moyenne des notes */
  double calculeMoyenne() {
    return ((double)notes) / nombreDeNotes ;
  }
}
```

UNIVERSITÉ CÔTE D'AZUR

Vocabulaire

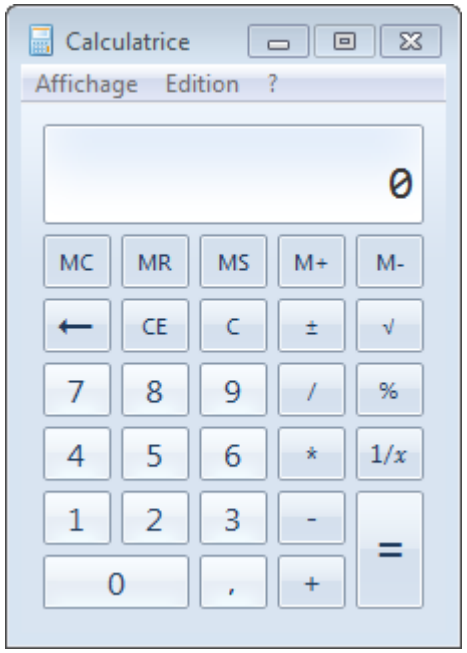


UNIVERSITÉ CÔTE D'AZUR

Calculatrice

- Calculatrice.java

```
class Calculatrice {
  double add(double v1, double v2) {
    return v1+v2;
  }
  double mul(double v1, double v2) {
    return v1*v2;
  }
  double inv(double v) {
    return 1/v;
  }
}
```



État : ?

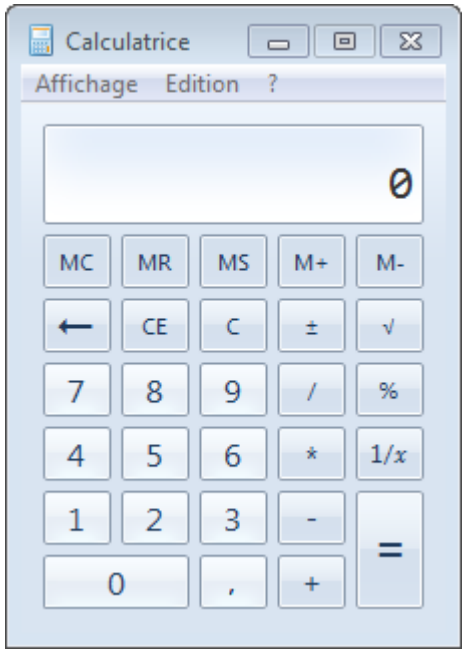
Usage: **Calculatrice c = new Calculatrice();**
c.add(12, 5);

UNIVERSITÉ CÔTE D'AZUR

Méthodes de classes : static

- Calculatrice.java

```
class Calculatrice {
  static double add(double v1, double v2) {
    return v1+v2;
  }
  static double mul(double v1, double v2) {
    return v1*v2;
  }
  static double inv(double v) {
    return 1/v;
  }
}
```



État : NON

Usage:

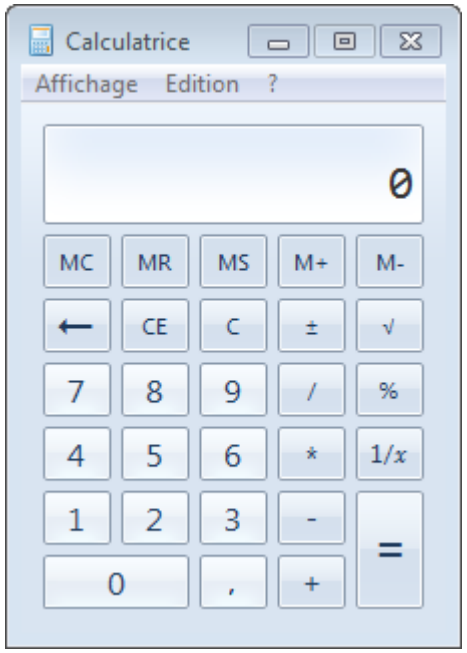
Calculatrice.add(12, 5);

UNIVERSITÉ CÔTE D'AZUR

Champs de classe : static

- Calculatrice.java

```
class Calculatrice {
  static double add(double v1, double v2) {
    return v1+v2;
  }
  static double mul(double v1, double v2) {
    return v1*v2;
  }
  static double inv(double v) {
    return 1/v;
  }
  static double PI = 3.1415;
}
```



État : NON

Usage:

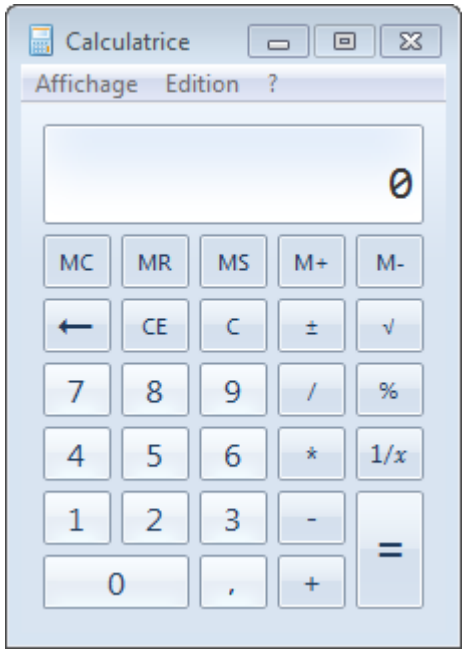
```
Calculatrice.add(12, 5);
Calculatrice.add(Calculatrice.PI, 5);
```

UNIVERSITÉ CÔTE D'AZUR

Constantes : final

- Calculatrice.java

```
class Calculatrice {
  static double add(double v1, double v2) {
    return v1+v2;
  }
  static double mul(double v1, double v2) {
    return v1*v2;
  }
  static double inv(double v) {
    return 1/v;
  }
  static final double PI = 3.1415;
}
```



État : NON

Accumulateur ?
Mémoire ?

Usage:

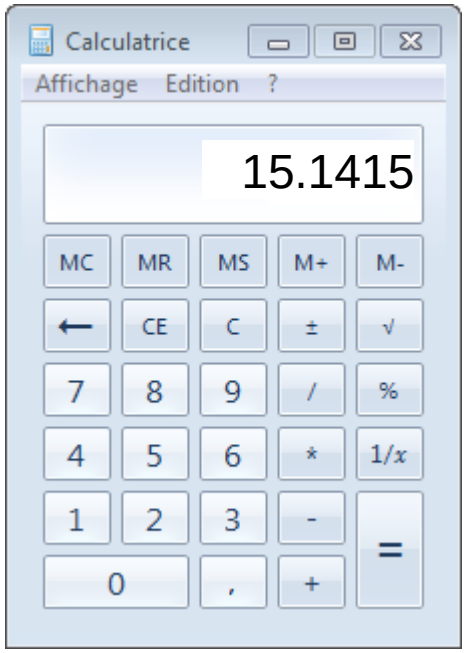
```
Calculatrice.add(12, 5);
Calculatrice.add(Calculatrice.PI, 5);
```

UNIVERSITÉ CÔTE D'AZUR

Méthodes d'instance et état

- Calculatrice.java

```
class Calculatrice {
  double accumulateur;
  void add(double v) {
    accumulateur += v;
  }
  void mul(double v) {
    accumulateur *= v;
  }
  void inv() {
    accumulateur = 1/accumulateur;
  }
  static final double PI = 3.1415;
}
```



État : OUI

Usage:
```
Calculatrice c = new Calculatrice();
c.add(12);
c.add(Calculatrice.PI);
```

UNIVERSITÉ CÔTE D'AZUR

« Regrouper DES objets de type différent par UNE classe qui rassemble les propriétés et opérations communes »

UNIFICATION

UNIVERSITÉ CÔTE D'AZUR

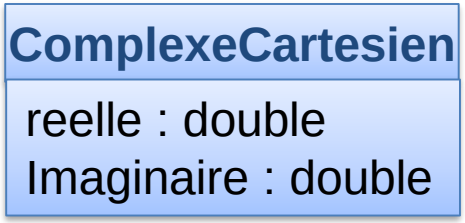
Les nombres complexes

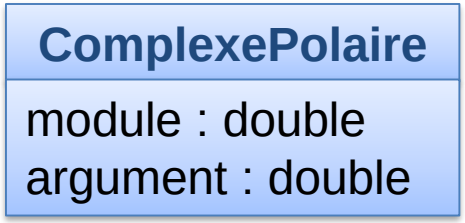
Codage cartésien

```
class ComplexeCartesien {
   double reelle;
   double imaginaire;
}
```

Codage polaire

```
class ComplexePolaire {
   double module;
   double argument;
}
```





Addition complexe:

(re1+i.im1) + (r2 + i.im2) = (re1+re2) + i.(im1+im2)

UNIVERSITÉ CÔTE D'AZUR

Addition complexe

- Addition en Java

```
static ComplexeCartesien addition(ComplexeCartesien c1,
                                  ComplexeCartesien c2) {
  double reelle = c1.reelle + c2.reelle;
  double imaginaire = c1.imaginaire + c2.imaginaire;
  return new ComplexeCartesien(reelle, imaginaire);
}

static ComplexeCartesien addition(ComplexePolaire c1,
                                  ComplexePolaire c2) {
  double reelle = c1.reelle() + c2.reelle();
  double imaginaire = c1.imaginaire() + c2.imaginaire();
  return new ComplexeCartesien(reelle, imaginaire);
}
```

Addition complexe:

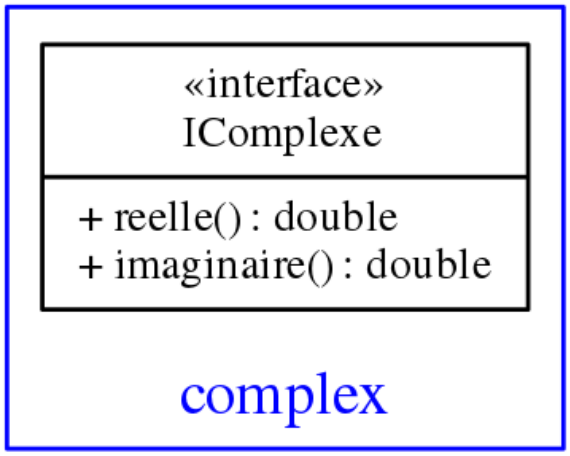
(re1+i.im1) + (r2 + i.im2) = (re1+re2) + i.(im1+im2)

UNIVERSITÉ CÔTE D'AZUR

Les interfaces

- Addition en Java

```
static ComplexeCartesien addition(IComplexe c1,
                                  IComplexe c2) {
  double reelle = c1.reelle() + c2.reelle();
  double imaginaire = c1.imaginaire() + c2.imaginaire();
  return new ComplexeCartesien(reelle, imaginaire);
}
```




```
interface IComplexe {
  double reelle();
  double imaginaire();
}
```

Addition complexe:

(re1+i.im1) + (r2 + i.im2) = (re1+re2) + i.(im1+im2)

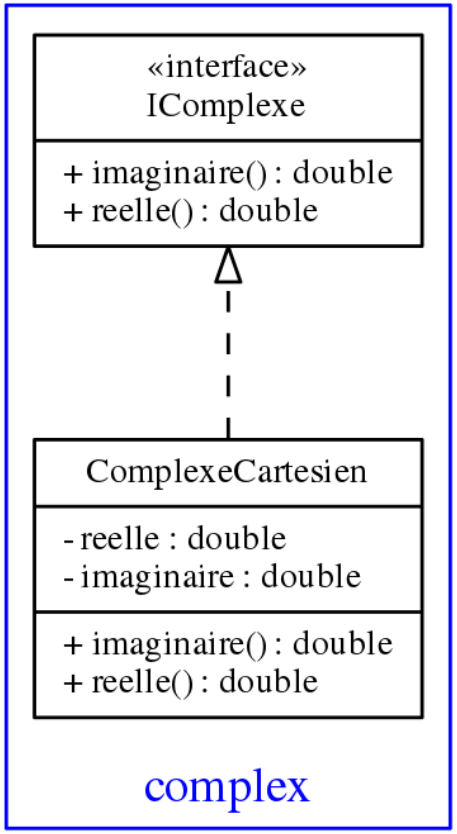
UNIVERSITÉ CÔTE D'AZUR

Implanter une interface

- ComplexeCartesien **est-un** IComplexe

```
class ComplexeCartesien implements IComplexe {
  private double reelle;
  private double imaginaire;

  public double reelle() { return reelle; }

  public double imaginaire() { return imaginaire; }
}
```

```
interface IComplexe {
  double reelle();
  double imaginaire();
}
```



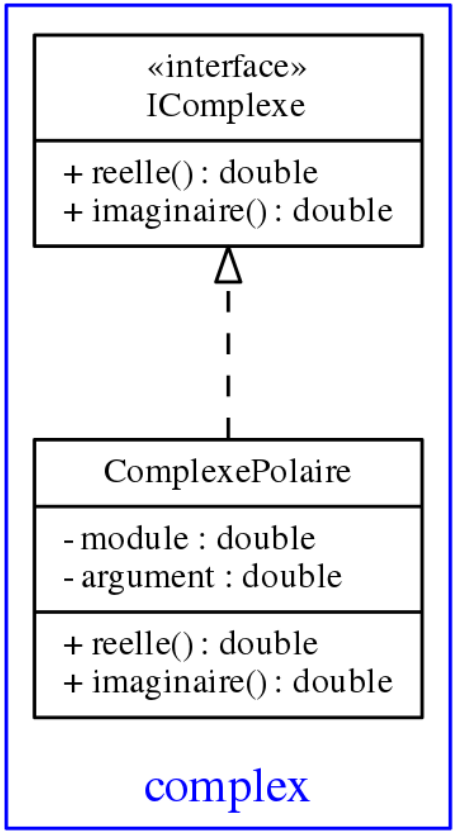
UNIVERSITÉ CÔTE D'AZUR

Implanter une interface

- ComplexePolaire **est-un** IComplexe

```
class ComplexePolaire implements IComplexe {
  private double module, argument;
  public double reelle() {
    return module*module*Math.cos(argument);
  }
  public double imaginaire() {
    return module*module*Math.sin(argument);
  }
}
```

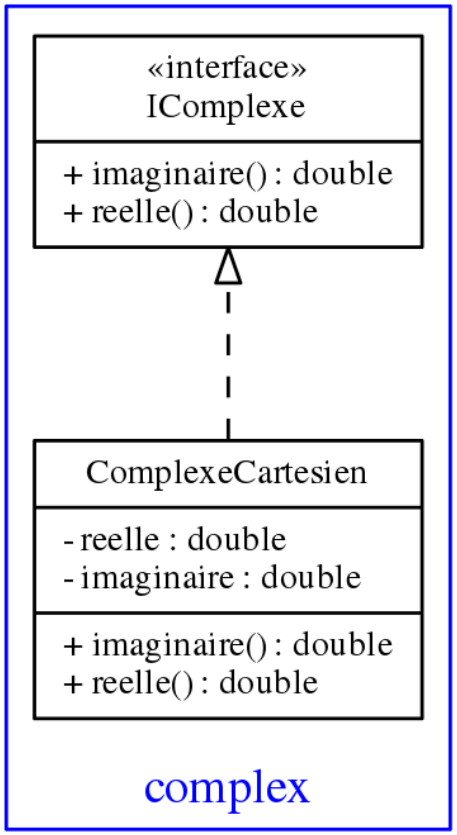
```
interface IComplexe {
  double reelle();
  double imaginaire();
}
```



UNIVERSITÉ CÔTE D'AZUR

Polymorphisme

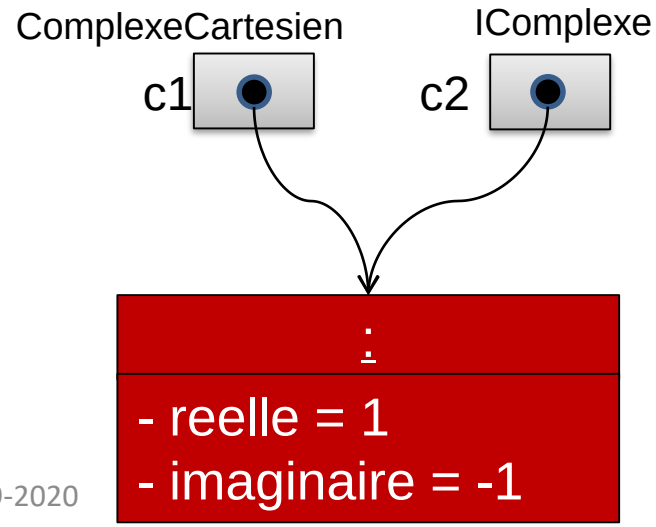
- Les références pointent des objets de **plusieurs** types
  - `IComplexe c1 = new ComplexeCartesien(1, -1);`
  - c1 est de **type statique** `IComplexe`
  - c1 référence un objet de **type dynamique** `ComplexeCartesien`
- Le type **statique**
  - Est choisi lors de la déclaration des références
- Le type **dynamique**
  - est choisi lors de l'affectation

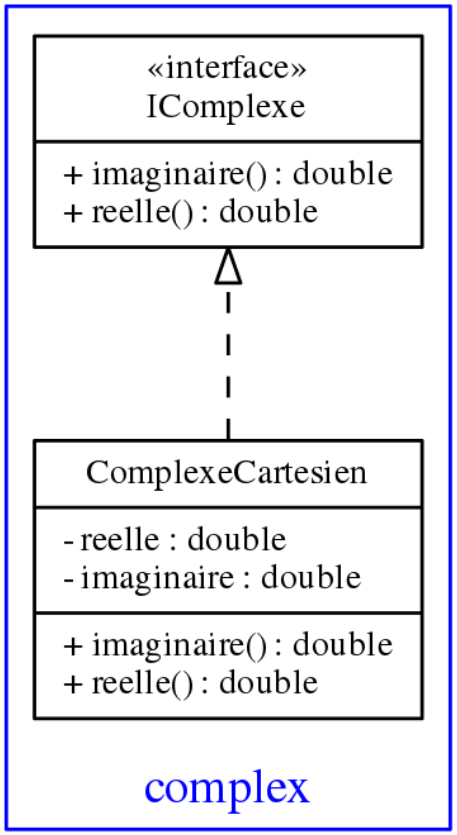


UNIVERSITÉ CÔTE D'AZUR

Polymorphisme

- Les références pointent des objets de **plusieurs** types
  - `ComplexeCartesien c1 = new ComplexeCartesien(1, -1);`
  - `IComplexe c2 = c1;`
- Le type **statique** sert à la compilation
  - c1.reelle est autorisé par le compilateur
  - ~~c2.reelle~~ est interdit par le compilateur
  - c1.reelle() est autorisé par le compilateur
  - c2.reelle() est autorisé par le compilateur



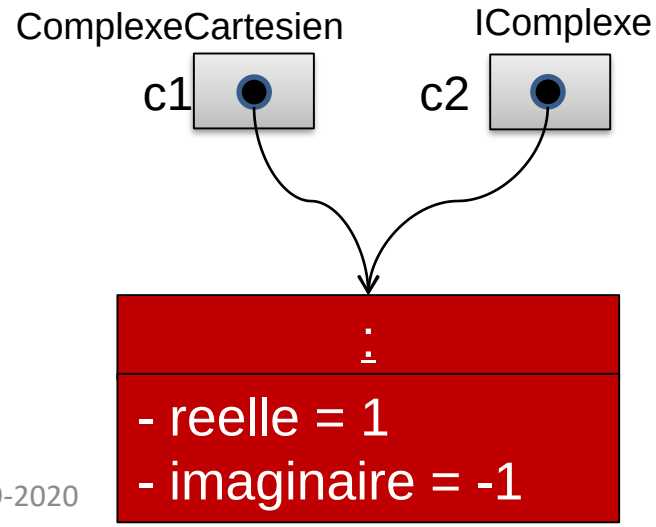


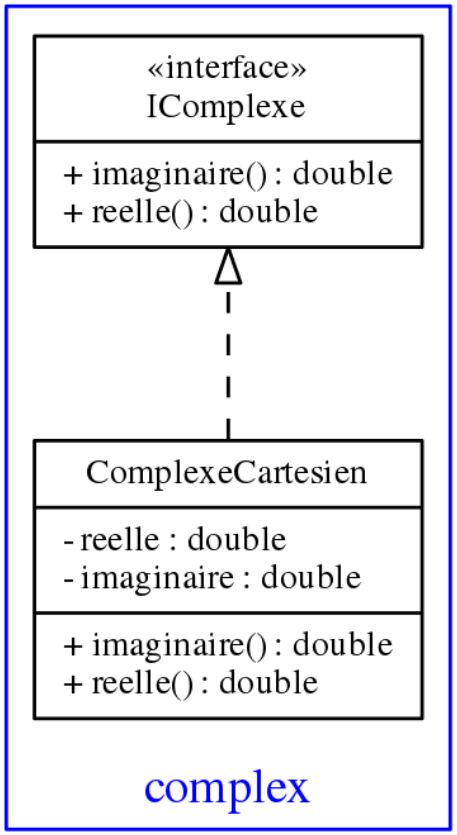
UNIVERSITÉ CÔTE D'AZUR

Liaison dynamique

- Les références pointent des objets de **plusieurs** types
  - `ComplexeCartesien c1 = new ComplexeCartesien(1, -1);`
  - `IComplexe c2 = c1;`
- Le type **dynamique** sert à l'exécution
  - c1.reelle() et c2.reelle() provoque

  toutes les deux l'exécution de la méthode `réelle()` de la classe `ComplexeCartesien`





UNIVERSITÉ CÔTE D'AZUR

« Les classes peuvent représentées des types structurés, mais aussi des comportements »

ABSTRACTION

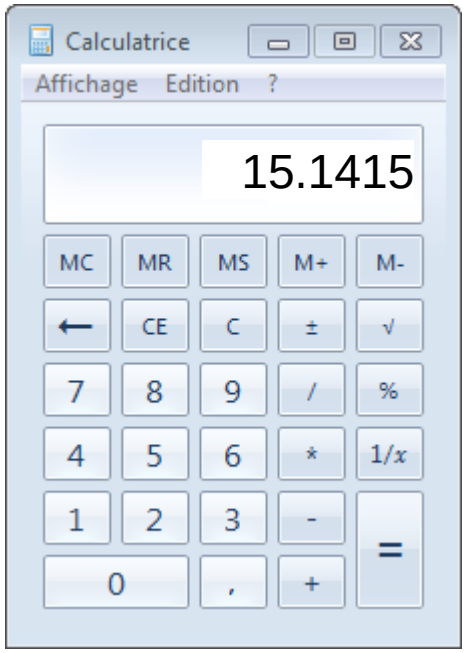
UNIVERSITÉ CÔTE D'AZUR

Méthodes d'instance et état

- Calculatrice.java

```
class Calculatrice {
  double accumulateur;
  void add(double v) {
    accumulateur += v;
  }
  void mul(double v) {
    accumulateur *= v;
  }
  void inv() {
    accumulateur = 1/accumulateur;
  }
  static final double PI = 3.1415;
}
```

Usage:
```
Calculatrice c = new Calculatrice();
c.add(12);
c.add(Calculatrice.PI);
```



État : OUI

Ajout d'opérations ?

UNIVERSITÉ CÔTE D'AZUR

Les opérations binaires

- Qu'est-ce qu'une opération binaire réelle ?

```
interface IOperationBinaire {
  double calcule(double c1, double c2);
}
```

- L'addition est une opération binaire !

```
class Addition implements IOperationBinaire {
  public double calcule(double c1, double c2) {
    return c1 + c2;
  }
}
```

- Usage

```
IOperationBinaire op = new Addition();
double v = op.calcule(12, 5); => 17
```

UNIVERSITÉ CÔTE D'AZUR

Les opérations binaires

- Qu'est-ce qu'une opération binaire réelle ?

```
interface IOperationBinaire {
  double calcule(double c1, double c2);
}
```

- La multiplication est une opération binaire !

```
class Multiplication implements IOperationBinaire {
  public double calcule(double c1, double c2) {
    return c1 * c2;
  }
}
```

- Usage

```
IOperationBinaire op = new Multiplication();
double v = op.calcule(12, 5); => 60
```

UNIVERSITÉ CÔTE D'AZUR

Méthodes d'instance et état

- Calculatrice.java

```
class Calculatrice {
  double accumulateur;
  void applique(IOperationBinaire op, double v) {
    accumulateur = op.calcule(accumulateur, v);
  }
  double inv() {
    accumulateur = 1/accumulateur;
  }
  static final double PI = 3.1415;
}
```

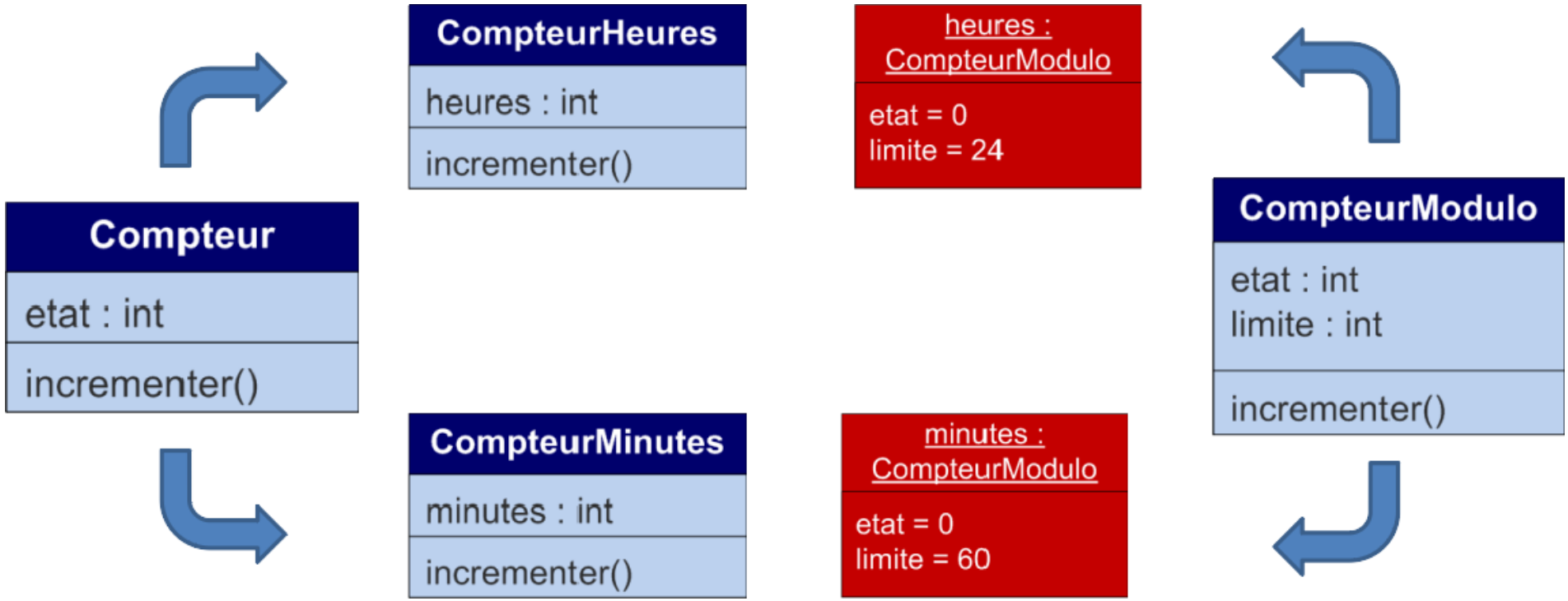
Usage:

```
Calculatrice c = new Calculatrice();
c.applique(new Addition(), 12);          c.add(12);
c.applique(new Addition(), 14);          c.add(14);
c.applique(new Multiplication(), 25);    c.mul(25);
```

UNIVERSITÉ CÔTE D'AZUR

Abstraction et réutilisation

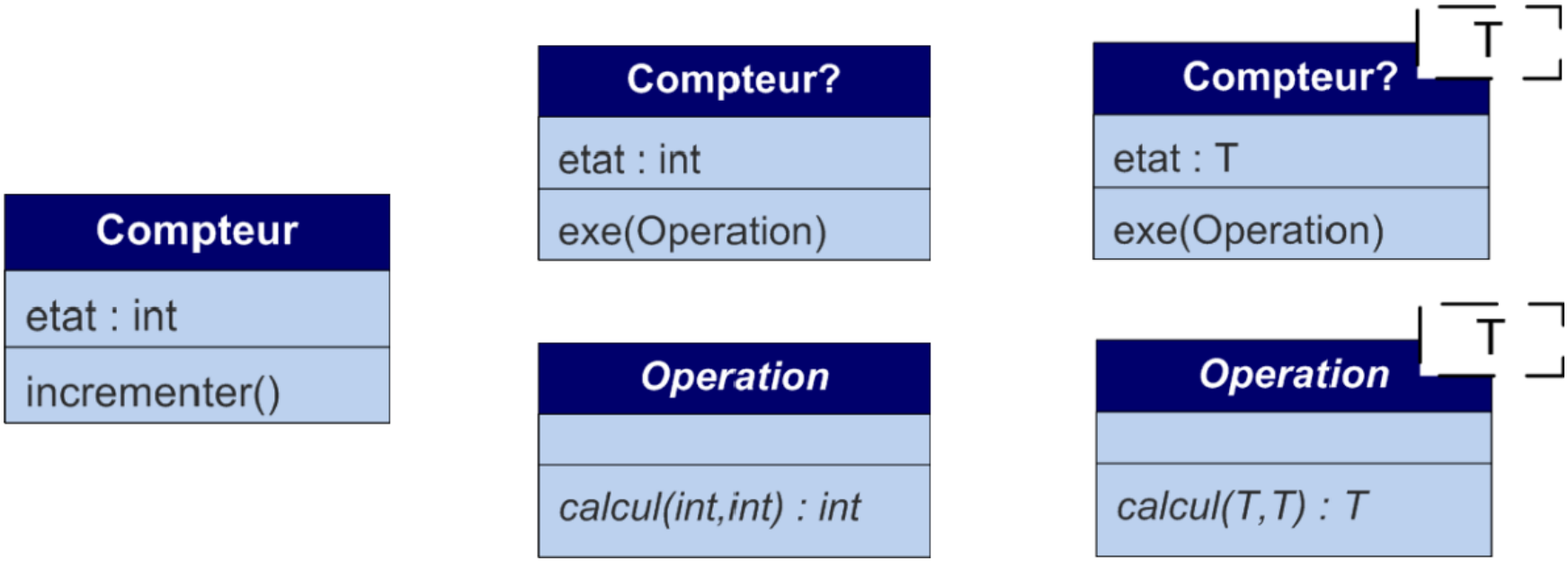
- Quelle est la limite à l'abstraction ?
  - Les classes deviennent des objets



UNIVERSITÉ CÔTE D'AZUR

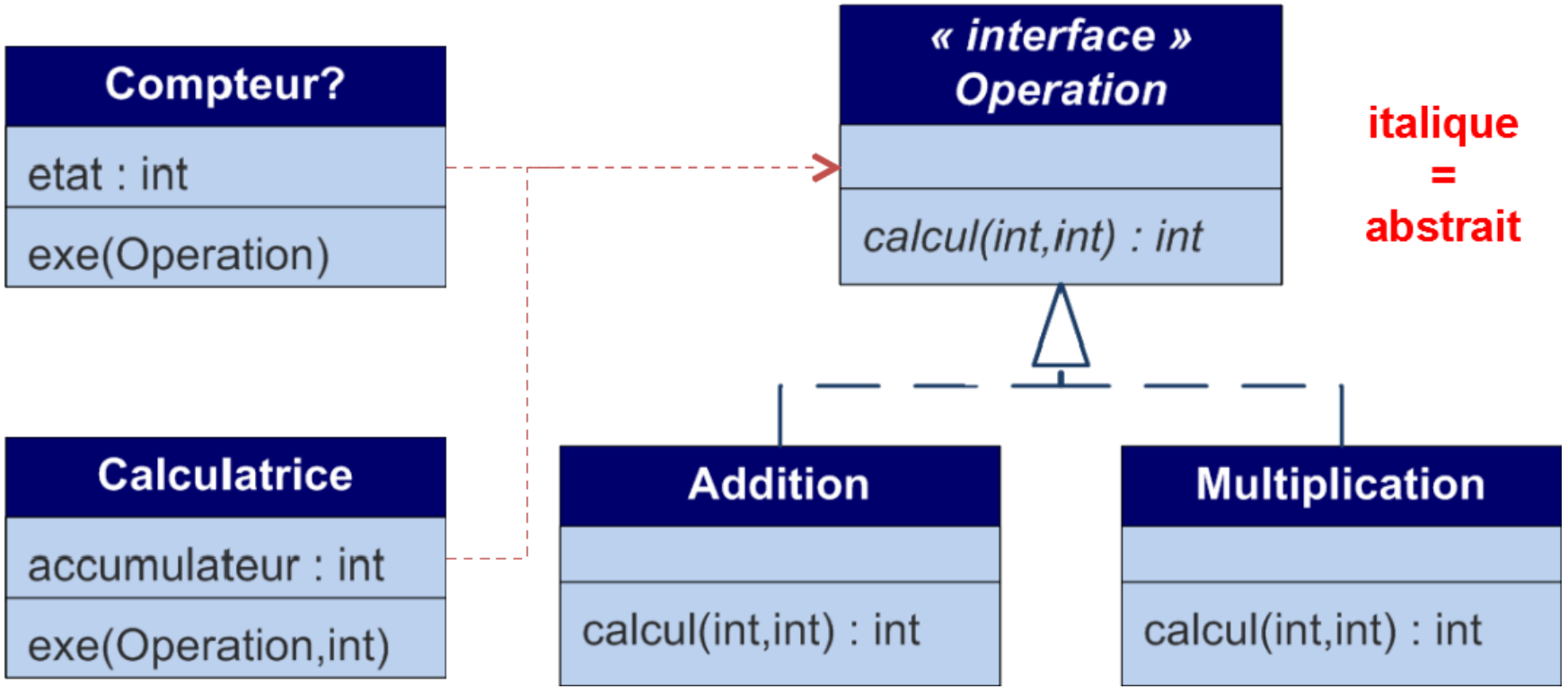
Abstraction et réutilisation

- Quelle est la limite à l'abstraction ?
  - Les méthodes deviennent des classes



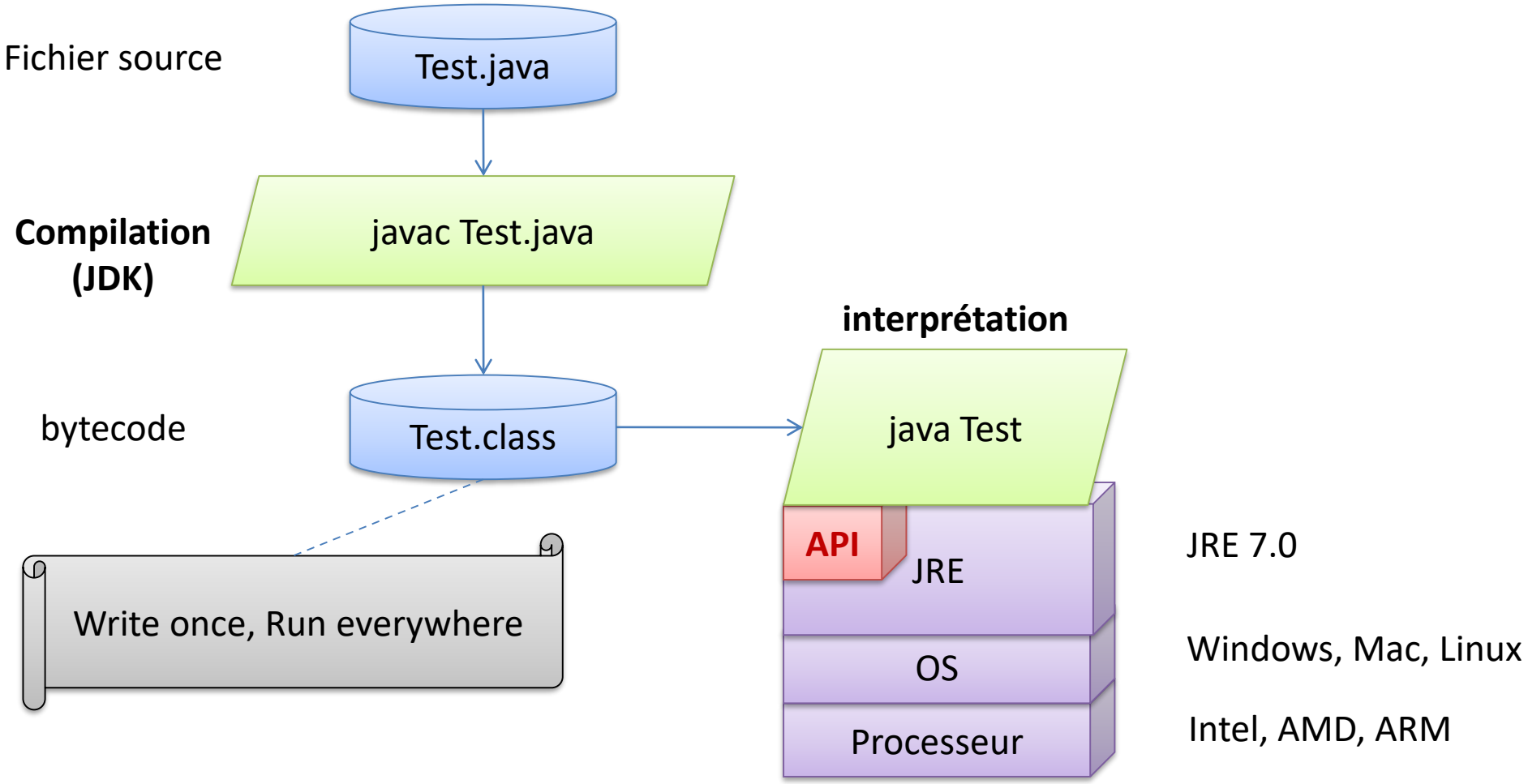
UNIVERSITÉ CÔTE D'AZUR

Abstraction vs. lisibilité



UNIVERSITÉ CÔTE D'AZUR

Processus de compilation en Java



UNIVERSITÉ CÔTE D'AZUR

Méthode main – java Test

Test.java:

```
Moyenne e = new Moyenne();

e.ajouteNote(8);
e.ajouteNote(18);
e.ajouteNote(12);

double m = e.calculeMoyenne();
```

Moyenne.java:

```
class Moyenne {
  int notes = 0 ;
  int nombreDeNotes = 0 ;

  void ajouteNote (int note) {
     notes += note;
     nombreDeNotes += 1;
  }

  double calculeMoyenne() {
    return ((double)notes) /
           nombreDeNotes ;
  }
}
```

UNIVERSITÉ CÔTE D'AZUR

Méthode main – java Test

Test.java:

```
class Test {

  static public void main(String[] args) {
    Moyenne e = new Moyenne();

    e.ajouteNote(8);
    e.ajouteNote(18);
    e.ajouteNote(12);

    double m = e.calculeMoyenne();

    System.out.println("Moyenne=" + m);
  }
}
```

Moyenne.java:

```
class Moyenne {
  int notes = 0 ;
  int nombreDeNotes = 0 ;

  void ajouteNote (int note) {
     notes += note;
     nombreDeNotes += 1;
  }

  double calculeMoyenne() {
    return ((double)notes) /
           nombreDeNotes ;
  }
}
```

UNIVERSITÉ CÔTE D'AZUR

Méthode main – java Test

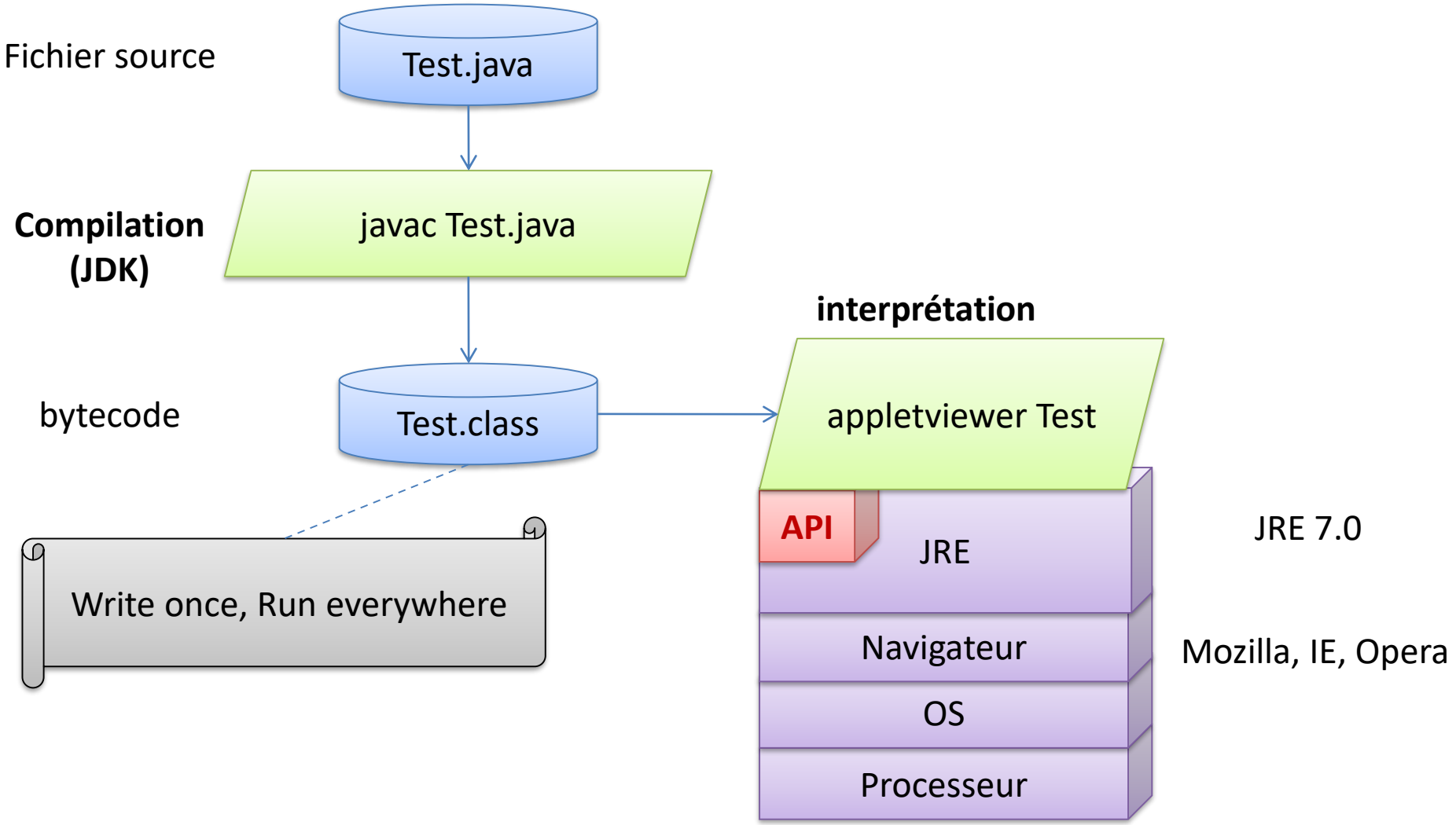
Test.java:

```
class Test {

  static public void main(String[] args) {
    Moyenne e = new Moyenne();

    e.ajouteNote(8);
    e.ajouteNote(18);
    e.ajouteNote(12);

    double m = e.calculeMoyenne();

    System.out.println("Moyenne=" + m);
  }
}
```

```
> dir
214 Moyenne.java
229 Test.java
> javac Test.java
> dir
411 Moyenne.class
214 Moyenne.java
744 Test.class
229 Test.java
>  java Test
Moyenne=12.666666666666666
```

UNIVERSITÉ CÔTE D'AZUR

Processus de compilation en Java



UNIVERSITÉ CÔTE D'AZUR

appletviewer Test.html

Test.java:

```
public class Test extends Applet {

  public void paint(Graphics g) {
    Moyenne e = new Moyenne();

    e.ajouteNote(8);
    e.ajouteNote(18);
    e.ajouteNote(12);

    double m = e.calculeMoyenne();

    g.drawString("Moyenne=" + m, 10, 10);
  }
}
```

Moyenne.java:

```
class Moyenne {
  int notes = 0 ;
  int nombreDeNotes = 0 ;

  void ajouteNote (int note) {
     notes += note;
     nombreDeNotes += 1;
  }

  double calculeMoyenne() {
    return ((double)notes) /
           nombreDeNotes ;
  }
}
```

UNIVERSITÉ CÔTE D'AZUR

appletviewer Test.html

Test.java:

```
public class Test extends Applet {

  public void paint(Graphics g) {
    Moyenne e = new Moyenne();

    e.ajouteNote(8);
    e.ajouteNote(18);
    e.ajouteNote(12);

    double m = e.calculeMoyenne();

    g.drawString("Moyenne=" + m, 10, 10);
  }
}
```

Test.html:

```
<object width="200" height="100">
  <param name="code"
         value="Test.class">
</object>
```

UNIVERSITÉ CÔTE D'AZUR

appletviewer Test.html

Test.java:

```
public class Test extends Applet {

  public void paint(Graphics g) {
    Moyenne e = new Moyenne();

    e.ajouteNote(8);
    e.ajouteNote(18);
    e.ajouteNote(12);

    double m = e.calculeMoyenne();

    g.drawString("Moyenne=" + m, 10, 10);
  }
}
```

Test.html:

```
<object width="200" height="100">
  <param name="code"
         value="Test.class">
</object>
```

```
> javac Test.java
> dir
411 Moyenne.class
214 Moyenne.java
702 Test.class
 92 Test.html
271 Test.java
>  appletviewer Test.html
```