Programmation objet avec le langage Java

Grenoble-INP Esisar

CS312

Objectifs du cours

- Compétences recherchées
  - Etre capable, à partir d'une spécification précise, de programmer une application en utilisant l'approche objet et le langage Java.
  - Etre capable d'évoluer dans un environnement de programmation objet.
  - Etre capable de concevoir une application objet simple.

Pourquoi ce cours?

- Le concept « objet » est très utile pour la modélisation/conception d'applications.
- L'approche « objet » est très courante et demandée dans l'industrie.
- Java est un des langages les plus utilisés aujourd'hui.
- Cela permet, également, de poursuivre la pratique de la programmation.

Equipe pédagogique et déroulement du cours

- 6 cours en amphi (1h30)
  - Enseignant : Ioannis Parissis
- 3 TD (1h30)
  - Enseignante : Nadine Marcos
- 8 TD machine/TP (1h30)
  - Enseignants : Guillaume Besset, Nadine Marcos

Evaluation

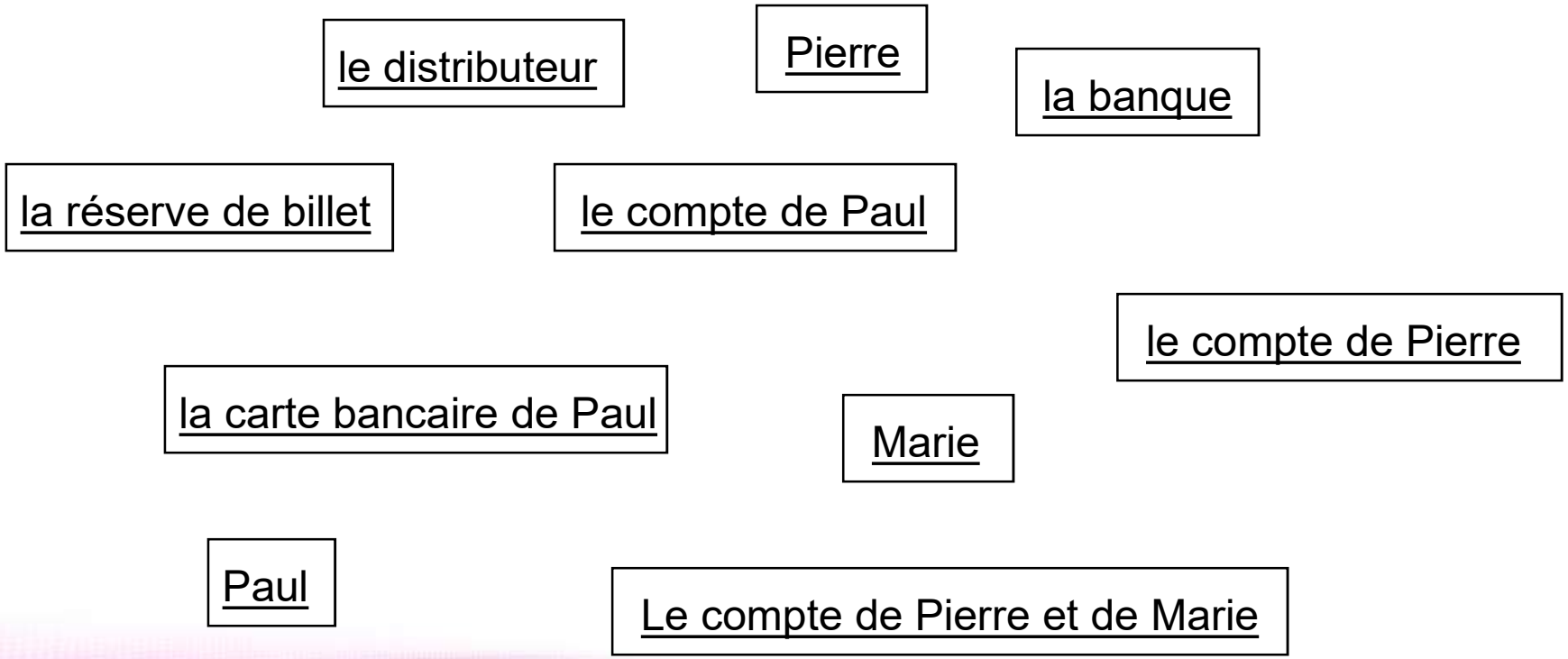
- Examen final écrit (60%)
- Notes de TP (20%)
- Note de contrôle continu (20%) – rendus TD

I. Introduction à l'approche objet

Les supports de ce chapitre sont en grande partie produits par Jean-Marie Favre, enseignant-chercheur UGA

Objets

Un système peut être vu comme un ensemble d'objets



Attributs et méthodes

Chaque objet

- a un **état** caractérisé par la valeur de ses **attributs**
- propose des **services** sous la forme de **méthodes**

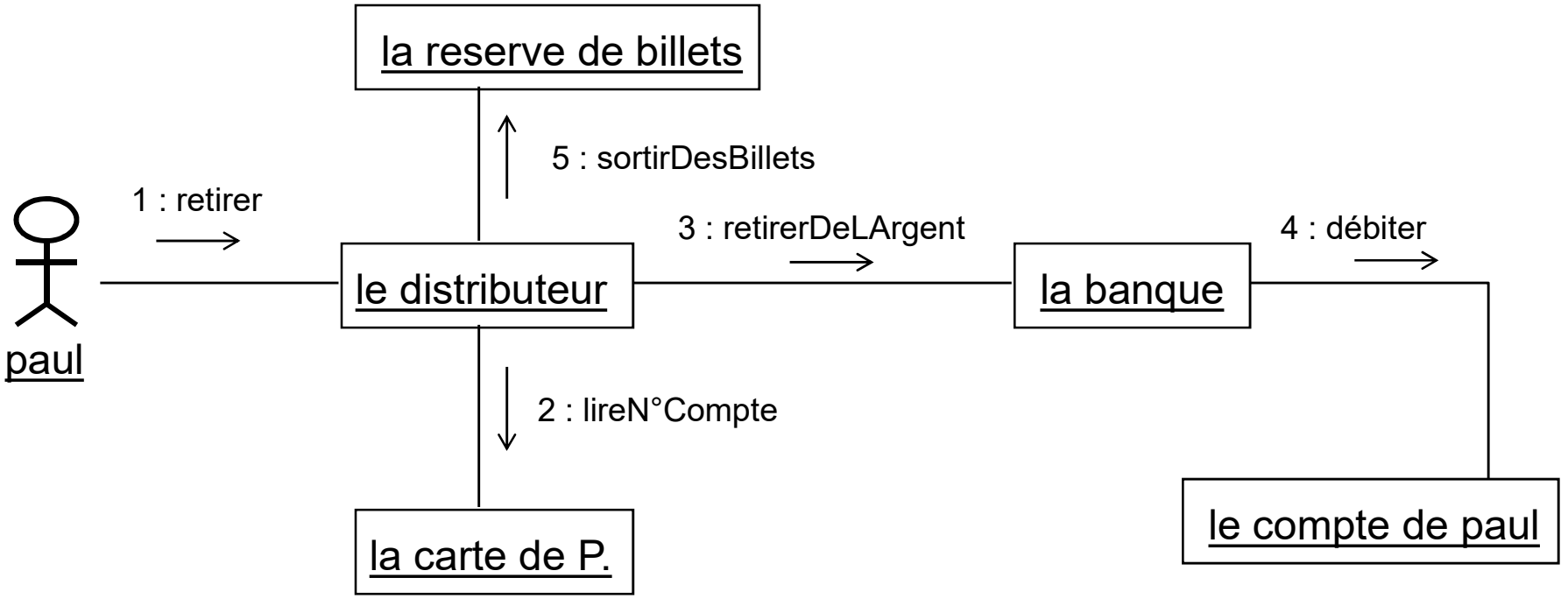
la banque
numéro = 2453 nom = « banque du sud » ...
Créer un compte Supprimer un compte Faire un virement Retirer de l 'argent ...

le compte de Paul
numéro = 88219 solde = 2000 découvert max = -500
ConsulterSolde Créditer Débiter

le compte de Pierre
numéro = 88213 solde = 10 découvert max = -100
ConsulterSolde Créditer Débiter

Envois de messages

Les objets interagissent par des **envois de messages**
(*appels de méthodes*)



Encapsulation

- Chaque objet indique les méthodes qu'il expose (son **interface**)
  ex: on sait que le distributeur permet de retirer de l'argent
- … mais cache la **réalisation concrète** des méthodes
  ex: on ne sait pas ce que fait le distributeur de manière interne lorsqu'il reçoit le message retirer de l'argent

la banque
numéro = 2453 nom = « banque du sud » …
Créer un compte Supprimer un compte Faire un virement Retirer de l 'argent …

le compte de Paul
numéro = 88219 solde = 2000 découvert max = -500
ConsulterSolde Créditer Débiter

le compte de Pierre
numéro = 88213 solde = 10 découvert max = -100
ConsulterSolde Créditer Débiter

Encapsulation

- Un objet n'expose jamais son état (attributs) directement

  ex: on ne sait pas si le distributeur mémorise le nombre d'opérations effectuées, la date de la dernière opération, etc.

- … sauf via des méthodes

  ex: on peut *consulter* le solde d 'un compte, *créditer* ou *débiter* un compte, mais pas modifier directement l'attribut *solde*

la banque
numéro = 2453 nom = « banque du sud » ...
Créer un compte Supprimer un compte Faire un virement Retirer de l 'argent ...

le compte de Paul
numéro = 88219 solde = 2000 découvert max = -500
ConsulterSolde Créditer Débiter

le compte de Pierre
numéro = 88213 solde = 10 découvert max = -100
ConsulterSolde Créditer Débiter

Avantages de l'encapsulation

- Un « client » ne connaît que l'interface de l'objet
  - l'objet est plus simple à comprendre
    ex: pas la peine de comprendre le contenu d'un lecteur DVD pour pouvoir l'utiliser.
  - l'objet plus simple à réutiliser
    ex: si l'interface est bien définie on peut le brancher à d'autres appareils (chaîne hi-fi, télévision…)
  - l'objet est protégé contre les mauvaises utilisations
    ex: on ne peut pas mettre les doigts dans le lecteur ni enlever le disque pendant que le moteur tourne...

Avantages de l'encapsulation

- La réalisation de l'objet peut être modifiée sans impact sur son interface avec le client

  – on peut améliorer l'objet et le faire évoluer

    ex: mettre un moteur plus puissant dans une voiture, remplacer un composant par un autre équivalent

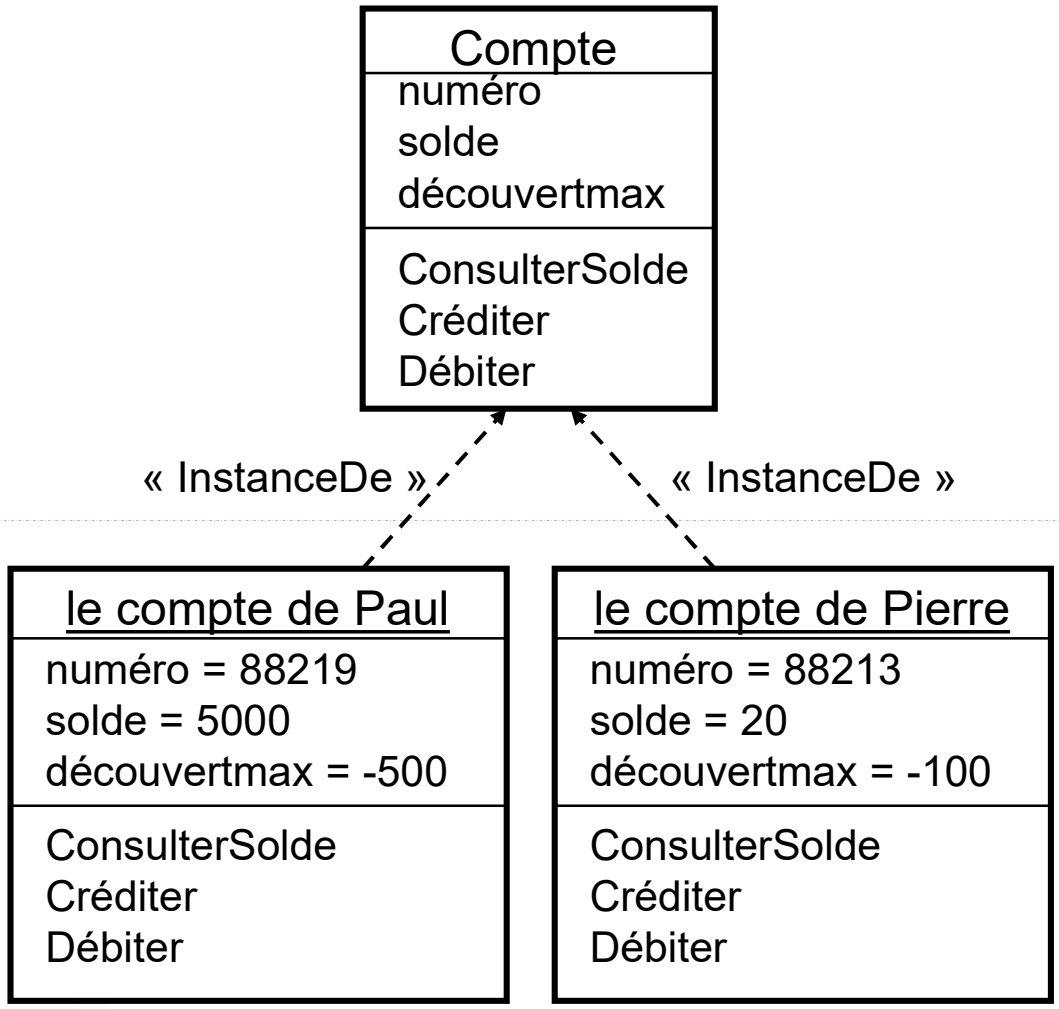
  – on pourrait remplacer l'objet par un autre équivalent

    ex: pas de problème pour passer d'une voiture à une autre

Classification

- On regroupe des objets similaires en **classes**
  - Une classe est un modèle, un «moule»
- Une classe est caractérisée par
  - ses attributs
  - ses méthodes
- Chaque objet est une **instance** d'une classe
  - Une classe permet d'**instancier** plusieurs objets
  - Chaque objet instancié a les attributs et méthodes de la classe

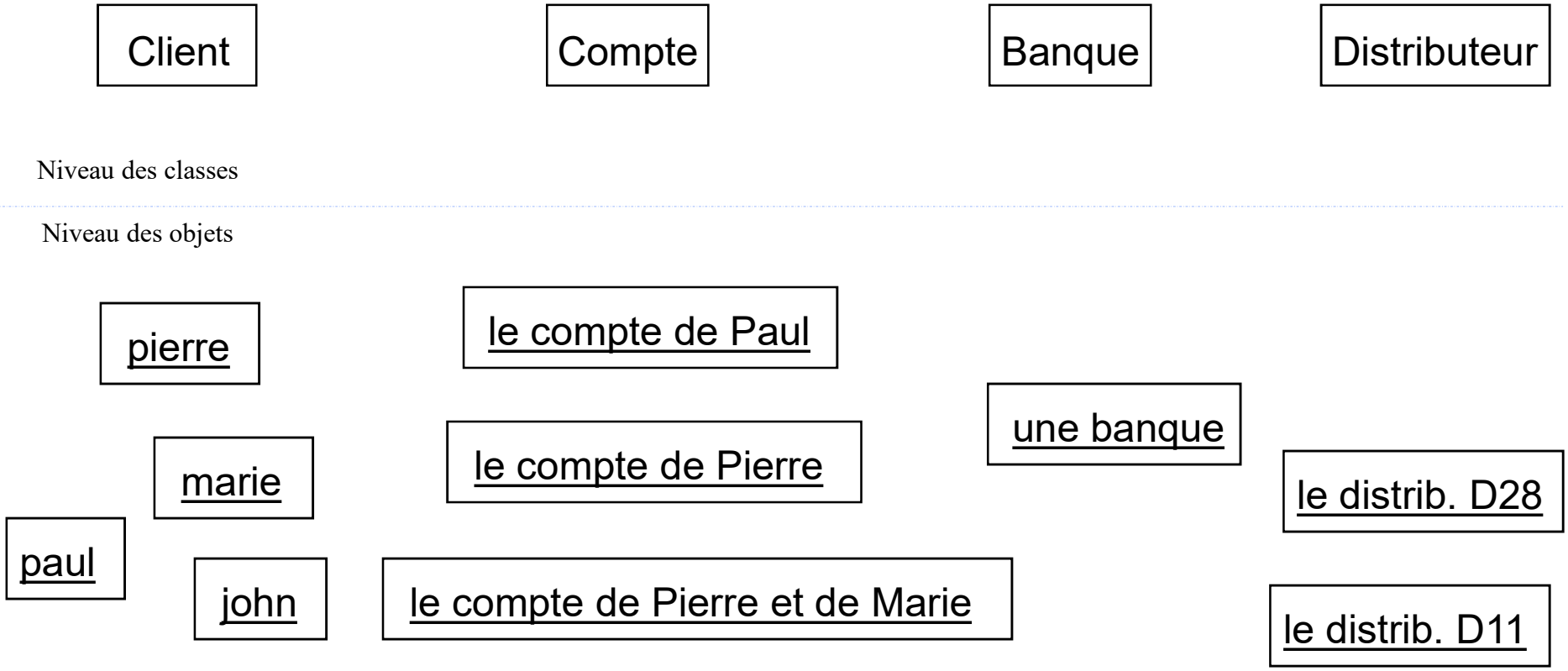
Classes et Objets



Niveau des classes (modèle)

Niveau des objets (instances)

Exemple de classification



Principe de « réification »

- **Réification**
  - Matérialiser un concept par un objet
- Un concept abstrait peut être « réifié »
  - l'événement « à 10h45 une carte bleue à été introduite »
- Une relation entre deux objets peut être « réifiée »
  - « jean *possède* la voiture immatriculée CS 312 JA »
    est réifié dans le monde réel par une carte grise
- Réifier un concept permet de le manipuler concrètement
- Question (ouverte): quels concepts réifier ?
  - Difficile de définir une « bonne » représentation en termes d'objets

Rappel des concepts introduits

- Objet
- Attribut
- Méthode
- Classe

- Réification
- Encapsulation
- Classification

II. Introduction au langage Java

II.1 Objets et classes en Java

Un exemple introductif

Exemple : un polygone régulier (en C)

- On souhaite réaliser un module permettant de représenter un polygone régulier
- Deux fonctions:
  - void initialiser (int nbDeCotes, int lgCote…)
  - int périmètre(…)
- Ecrire ce module en C
  - Définir une **structure de données**
  - Réaliser les **fonctions**
  - *Voir polygoneRegulier.h, polygoneRegulier.c*

Exemple : un polygone régulier (en C)

polygoneRegulier.h

```
struct structPoly{
    int nbDeCotes ;
    int lgCote ;
};

typedef struct structPoly Polygone;

void initialiser (Polygone*, int, int);
int perimetre(Polygone);
```

polygoneRegulier.c

```
void initialiser(Polygone* p, int nbDeCotes, int lgCote){
    if(nbDeCotes > 2){
        p->nbDeCotes = nbDeCotes ;
     }
    else {
           p->nbDeCotes = 3 ;
    }
    p->lgCote = lgCote ;
}

int perimetre(Polygone p){
        return p.nbDeCotes * p.lgCote ;
}
```

Une bonne utilisation

main.c

```
#include "polygoneRegulier.h"

int main()
{
    Polygone p;
    int per;

    initialiser(&p, 4, 5);
    per = perimetre(p);

    printf("Le perimetre est : %d", per);

    return 0;
}
```

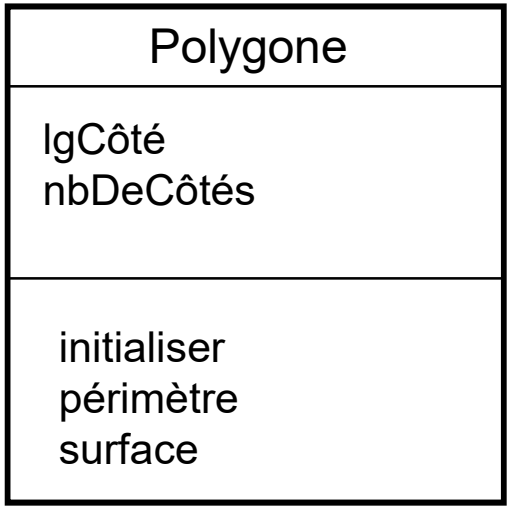
… et une mauvaise

main.c

```
#include "polygoneRegulier.h"

int main()
{
    Polygone p;
    int per;

    initialiser(&p, 4, 5);

    p.nbDeCotes = 2;

    per = perimetre(p);

    printf("Le perimetre est : %d", per);

    return 0;
}

/* p peut être modifié de manière inappropriée par une fonction "utilisatrice"
   (ici : main) */

/* La représentation du polygone par deux entiers est visible par les fonctions
   utilisatrices */
```

Réalisation en Java

```
public class Polygone {
  private int lgCôté;
  private int nbDeCôtés;

  public void initialiser(int nombre, int longueur){
    lgCôté = longueur;
    if (nombre > 2) nbDeCôtés = nombre;
    else nbDeCôtés = 3;
  }

  public int périmètre(){
   return lgCôté * nbDeCôtés ;
  }

}
```



Objets et classes en Java

- Un programme informatique « orienté objet » correspond à un ensemble d'objets représentant une partie du monde
- Les objets manipulés par le programme sont créés à partir de classes
  - Plusieurs objets similaires peuvent être créés à partir de la même classe
  - Chaque objet dispose d'un état (valeurs des champs – ou attributs)
- Une fois créé, un objet peut recevoir des messages *via* l'appel de méthodes
  - Les méthodes peuvent disposer de paramètres
  - Un objet peut appeler des méthodes d'un autre objet
- ***Écrire un programme Java = décrire des classes***
  - ***vs Ecrire un programme en C (= décrire des structures des données et des algorithmes les manipulant)***

Exemple de classe Java

```
public class Polygone{
  …
 public void initialiser(int nombre, int longueur) { …..}

 public int périmètre() { …..}

}


// un usage de cette classe (dans le code d'une méthode d'une classe autre
   que Polygone)

...
Polygone unPolygone = new Polygone();
unPolygone.initialiser(4,100);
int y = unPolygone.périmètre();
...
```

Attributs (champs d'instance), état

```
public class Polygone{
   private int lgCôté;
   private int nbDeCôtés;
   ...
}
```

- Valeurs des champs de l'instance : **état de l'objet**

Attributs privés

```
public class Polygone{
   private int lgCôté;
   private int nbDeCôtés;

   public void initialiser(int
    nombre, int longueur){
     lgCôté = longueur;
      if (nombre > 2)
           nbDeCôtés = nombre;
     else nbDeCôtés = 3;
  }

  public int périmètre(){
    return lgCôté * nbDeCôtés ;
  }

}
```

- Les champs sont **privés (`private`)**:
  - ***ils ne sont visibles que depuis les méthodes de la classe* Polygone**
  - Ces utilisations sont **interdites:**

```
y = polygone.lgCôté

polygone.lgCôté = 5
```

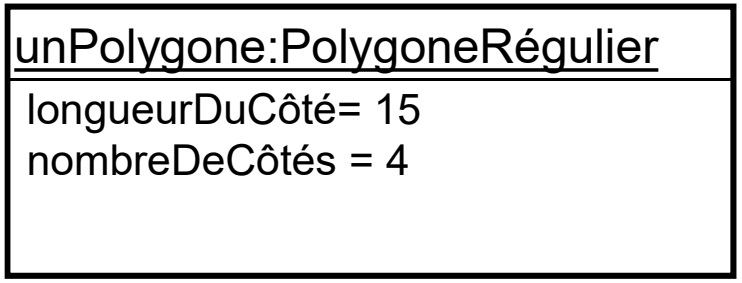
Signatures

```
public class Polygone{
   private int lgCôté;
   private int nbDeCôtés;

   public void initialiser(int nombre, int longueur){
        lgCôté = longueur;
        if (nombre > 2) nbDeCôtés = nombre;
        else nbDeCôtés = 3;
   }


  public int périmètre(){
   return lgCôté * nbDeCôtés ;
   }

}
```

- **Signature** d'une méthode : nom, type de retour et type des paramètres

Création d'objets: new

```
   ...
Polygone unPolygone = new  Polygone ();
unPolygone.initialiser(4,15);
   ...
```



Création d'objets
Opérateur new

```
...
  Polygone unPolygone = new Polygone();
  unPolygone.initialiser(4,15);
  Polygone autrePolygone = new Polygone();
  autrePolygone.initialiser(6,20);
  ...
```

unPolygone:PolygoneRégulier
longueurDuCôté= 15 nombreDeCôtés = 4

unPolygone:PolygoneRégulier
longueurDuCôté= 20 nombreDeCôtés = 6

Constructeurs

```
Polygone unPolygone = new Polygone();
unPolygone.initialiser(4,100);
```

- Il est plus naturel (et plus sûr) d'initialiser un objet dès sa création:

```
Polygone unPolygone = new Polygone(4,100);
```

- Pour cela, il faut que Polygone possède un « ***constructeur*** »

Constructeurs : exemple

```
public class Polygone {
  private int lgCôté;
  private int nbDeCôtés;

  public Polygone (int nombre, int longueur){
      lgCôté = longueur;
      if (nombre > 2) nbDeCôtés = nombre;
      else nbDeCôtés = 3;
  }
}

// Maintenant, on peut écrire :
...
Polygone unPolygone = new Polygone (4,100);


...
```

Méthodes « accesseurs » (« getters »)

- **Par convention** nommés **`public`** ***`type`*** `get`***`Nom`***
- Permettent de consulter l'état de l'objet

```
public class Polygone {
  private int lgCôté;
  private int nbDeCôtés;

  public int getNbDeCôtés(){
    return nbDeCôtés;
  }

  public int getLgCôté(){
    return lgCôté;
  }
}
```

Méthodes « mutateurs » (« setters »)

- **Par convention** **`public`** ***`void`*** `set`***`Nom (type id)`***
- Permettent de modifier l'état de l'objet

```
public class Polygone{
   private int lgCôté;
   private int nbDeCôtés;

   public void setLgCôté(int longueur){
      lgCôté = longueur;
   }

   public void setNbDeCôtés(int nombre){
      if (nombre > 2) nbDeCôtés = nombre;
   }
```

Accesseurs-Mutateurs

Attention: les attributs ne sont pas toujours ceux qu'on croit!

```
public class Polygone{
  private int nbDeCôtés;
  private int périmètre;

public Polygone (int nombre, int longueur){
  if (nombre > 2) nbDeCôtés = nombre;
  else nbDeCôtés = 3;
  périmètre = nbDeCôtés * longueur;
  }

  public int getLgCôté(){
    return(périmètre / nbDeCôtés);
  }

  public void setLgCôté(int longueur){
      périmètre = nbDeCôtés * longueur;
  }
}
```

Résumé des méthodes d'une classe

- **Constructeur**
  - Initialisation/configuration d'un objet à sa création
- **Accesseurs**
  - Consultation de l'état d'un objet
- **Mutateurs**
  - Modification de l'état d'un objet
- Autres méthodes

II. Introduction au langage Java

II.2 Expressions et types dans Java

Types de données

- Types primitifs

```
int x, y;
boolean b = false;
double d = 3.14159;
```

- Objets

```
Polygone unPolygone = new Polygone();
Carré unCarré = new Carré();
```

Types primitifs java

- entier
  - signés seulement
  - type byte (8 bits), short (16 bits), int (32 bits), long (64 bits)
- flottant
  - standard IEEE
  - type float (32 bits), double (64 bits)
- booléen
  - type boolean (true,false)
- caractère
  - unicode,
  - type char (16 bits) http://www.unicode.org

Conversions

- Automatique
  - si la taille du type destinataire est supérieure
  - byte a,b,c;
  - int d = a+b/c;

- Explicite
  - byte b = (byte)200;
  - 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

Déclarations de constantes

- `private final static double CAPACITE = 50.0;`
- `private final static String hi = "hello";`
- `public final int MAXIMUM = 100;`
- `public final long TAILLE_MAX = 100L;`
- `public final static byte MAX = (byte)0xFF;`

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 \u03FFF alphabet grec
  - .....
- http://www.unicode.org

Opérateurs

- Arithmétiques
  - +, -, *, /, %, ++ +=, -=, /=, %=, --,
- Binaires (bit à bit)
  - ~, &, |, ^ *(ou exclusif)*, &=, |=, ^=
  - >>, >>> *(décalage et remplissage avec 0)*, <<, >>=, >>>=, <<=
- Relationnels
  - ==, !=, >, <, >=, <= Syntaxe C
- Booléens
  - &, |, ^, &=, |=, ^=, ==, !=, !, ?:
  - ||, &&

Opérateurs booléens

```
1  public class Div0{
2    public void exempleDeCourtCircuits(
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)

Opérateurs : exemples

- `+ , - , * , /`

```
int x = 0;
int y = x + 1;
```

- `-=,+=`

```
int x = 2;  x += 2;  // x = x  + 2;
```

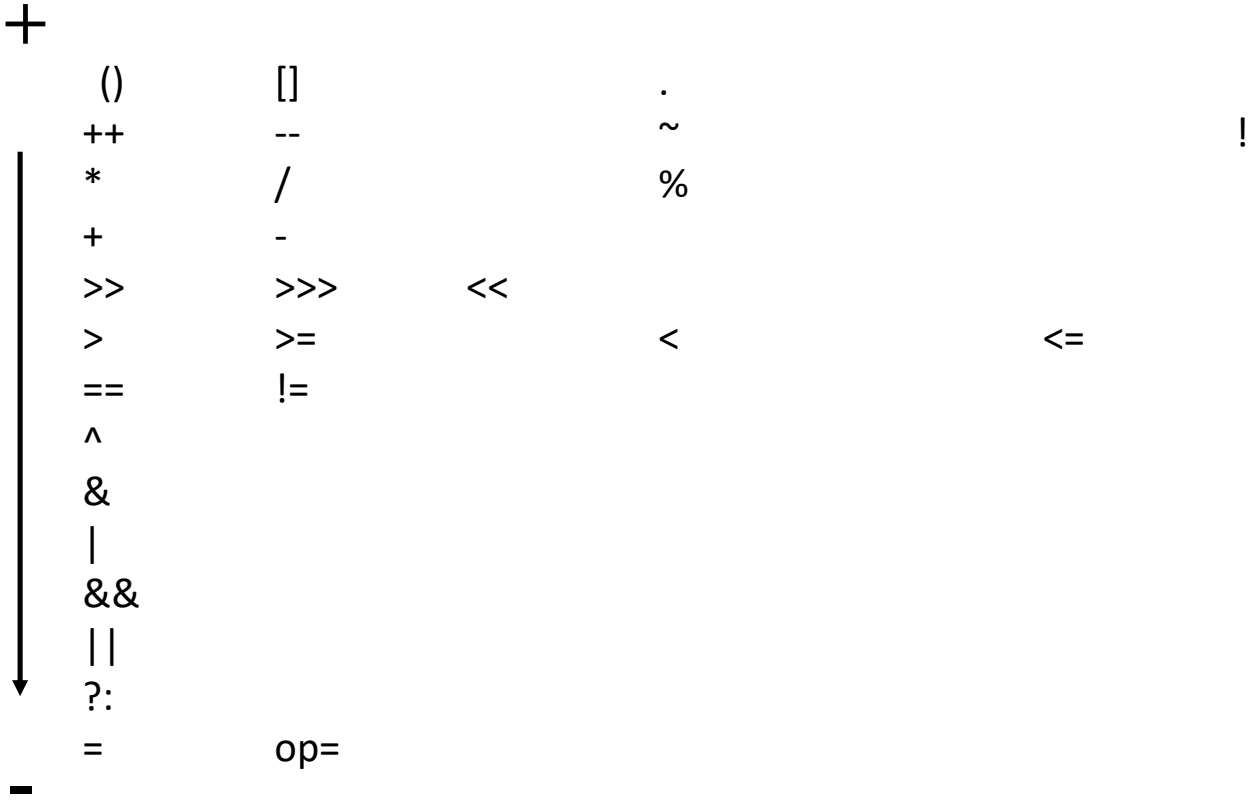
- `++, --`

```
x++;  // x = x + 1 ou x += 1
```

- `+`

```
String s = "bon"  + "jour";
int amount = 1000;
System.out.println("amount = " + amount);
```

Précédence des opérateurs (priorité)

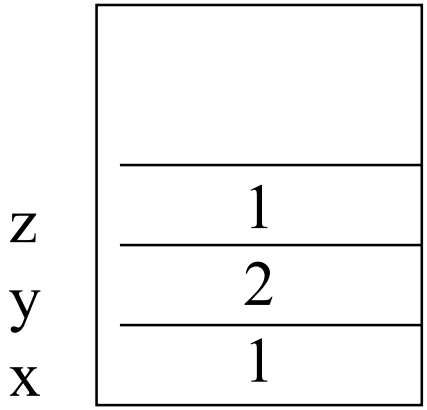


- int a = 1, b = 1, c=2;
- int x = a | 4 + c >> b & 7 | b >> a % 3; // ??? Que vaut x ???

Affectation : type primitif vs. objets

- Type primitif :

```
int    x = 1;
int    y = 2;
int    z = x;
```



- Object

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


o1 = o;
```

?

Exemple : quel résultat?

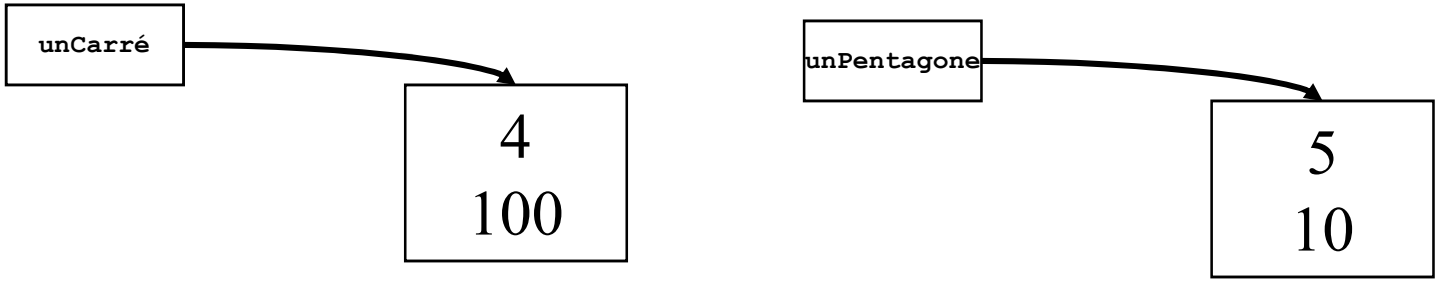
- ```
  int a;
  int b;
  a = 32;
  b = a;
  a = a + 1;
  System.out.println(b);
  ```

- ```
  Person a;
  Person b;
  a = new Person("Anakin");
  b = a;
  a.changeName("Darth Vader");
  System.out.println( b.getName());
  ```

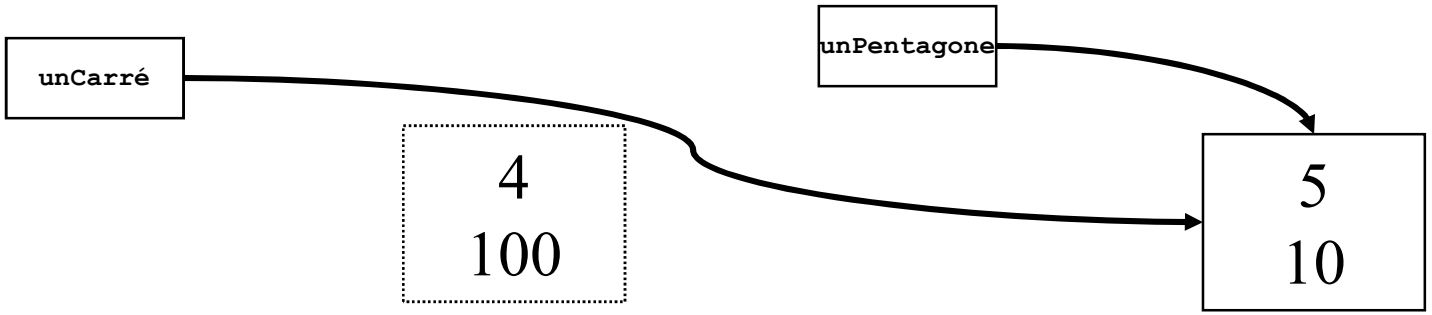
- `System.out.println(String) : méthode d'affichage`

Affectation d'objets

```
Polygone unCarré, unPentagone;
unCarré = new Polygone (4,100);
unPentagone = new Polygone (5,10);
```




```
unCarré = unPentagone;
```



Que devient cet objet?

cf. « ramasse-miettes »

Identité vs égalité d'objets

```
Polygone unCarré, unAutreCarré;
unCarré = new Polygone (4,100);
unAutreCarré = new Polygone (4,100);
unTroisièmeCarré = unCarré;

if(unCarre == unAutreCarre) {
    System.out.println("Les objets sont identiques");
}

if(unCarre.equals(unAutreCarre)) {
    System.out.println("Les objets sont égaux");
}

if(unTroisièmeCarre == unCarré) {
    System.out.println("Les objets sont identiques");
}
```

Portée et durée de vie des variables

```
public class Polygone {
  private int lgCôté;
  private int nbDeCôtés;

  public Polygone (int nombre, int longueur){
     int lgCôté = longueur;    // Attention!
     nbDeCôtés = nombre;
  }
}
```

this

```
public class Polygone {
   private int lgCôté;
   private int nbDeCôtés;

   public Polygone (int nombre, int lgCôté){
        lgCôté = lgCôté;
        nbDeCôtés = nombre;
   }
```

- Solution

```
public Polygone (int nombre, int lgCôté){
        this.lgCôté = lgCôté;
        nbDeCôtés = nombre;
   }
```

this désigne l'objet exécutant la méthode

this.lgCôté : champ lgCôté de l'objet exécutant la méthode etc..

Méthode main

```
public class Premice {

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

- Une seule méthode main dans une application (i.e. pour toutes les classes).

- La classe contenant cette méthode est dite "principale" (main).

Bilan des concepts introduits

- Classes et objets java
- Méthodes
  - Signature
  - Constructeurs, accesseurs, mutateurs, autres
- Types primitifs
  - Opérateurs, priorités
  - Conversion
- Type objet (Object)
  - Affectation
- Comparaison d'objets
  - Egalité, identité
- Portée - durée de vie des variables
  - Utilisation de `this`
- Affichage à l'écran
  - System.out.println