

# Initiation à la programmation Java (2)

## IP2

Yan Jurski

21 janvier 2021

- Les cours en amphi ne se dérouleront pas en présentiel
  - Notre proposition consistait pour IP2 et Math à scinder l'amphi en 2 ce qui explique la présence des 2 créneaux sur la version DSE des emplois du temps. Nous y reviendrons peut être en cours d'année...
- TDs et TPs commenceront la semaine prochaine
  - présentiel ou distanciel ? (éternelle question)
  - Les services du planning ont reçu un feu vert de dernière minute pour organiser les 2 semaines qui viennent en présentiel. Avec des conditions qui expliquent le retard dans les informations sur vos groupes.
  - Un mode dégradé est envisagé en plan B, sous la forme de demi-groupes de TD/TP. La présence serait alternée en combinaison avec un appoint distanciel.
  - Enfin, il faut toujours être prêt à ce que les enseignement se déroulent ponctuellement tous à distance (c'était le cas l'an dernier, le programme avait alors été allégé)

- Des conséquences au quotidien pour vous et vos études  
fatigue, transport, examens reportés, cours annulés, ...
- Et puis ce n'est pas fini ...  
chaque quinzaine : durcissement ? assouplissement ? et vice/versa
- Nous avons conscience des défis qui s'imposent à vous. Pour les affronter il vous faut vous mettre le plus possible dans de bonnes conditions :
  - s'imposer de suivre un rythme
  - organiser son temps
  - organiser son espace de travail
- L'ufr d'informatique peut prêter quelques machines
- L'université propose un accompagnement :  
<https://u-paris.fr/besoin-dune-aide-psychologique/>

# Organisation de ce cours d'IP2

- Amphi 2h
- TD 2h
- TP 2h

Ce n'est plus un cours/TD comme IP1 l'était au premier semestre !

Page moodle *IF12Y010 Initiation à la Programmation 2*

plusieur notes : en TD, TP, partiel, examen

- note de contrôle continu :  $\frac{1}{3} TP + \frac{2}{3} TD$
- note d'amphi :  $\max(\text{exam mai}, \frac{\text{partiel} + \text{exam mai}}{2})$
- note finale : 15% *contrôle* + 85% *amphi*
- rattrapage :  $\max(\text{juin}, 15\% \text{ contrôle} + 85\% \text{ juin})$

plusieur notes : en TD, TP, partiel, examen

- note de contrôle continu :  $\frac{1}{3} TP + \frac{2}{3} TD$
- note d'amphi :  $\max(\text{exam mai}, \frac{\text{partiel} + \text{exam mai}}{2})$
- note finale : 15% *contrôle* + 85% *amphi*
- rattrapage :  $\max(\text{juin}, 15\% \text{ contrôle} + 85\% \text{ juin})$

## Contrôle continu obligatoire

Absence de note en TD ou TP

⇒ pas de calcul de note finale

⇒ session de rattrapage

# Contrôle des connaissances

plusieur notes : en TD, TP, partiel, examen

- note de contrôle continu :  $\frac{1}{3} TP + \frac{2}{3} TD$
- note d'amphi :  $\max(\text{exam mai}, \frac{\text{partiel} + \text{exam mai}}{2})$
- note finale : 15% *contrôle* + 85% *amphi*
- rattrapage :  $\max(\text{juin}, 15\% \text{ contrôle} + 85\% \text{ juin})$

## Contrôle continu obligatoire

Absence de note en TD ou TP

⇒ pas de calcul de note finale

⇒ session de rattrapage

Sous réserve des conditions particulières à cette année

# Perspective de ce cours





## Automobile

- Conducteur : pilote, taxi ...

## Informatique

- Utilisateur

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...

## Informatique

- Utilisateur

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...
- Développeur - Intégrateur
  - **composition**
  - **manipulation concepts**

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte
- Concepteur des pièces
  - spécialiste
  - prêtes à être branchées

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...
- Développeur - Intégrateur
  - **composition**
  - **manipulation concepts**

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte
- Concepteur des pièces
  - spécialiste
  - prêtes à être branchées

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...
- Développeur - Intégrateur
  - composition
  - manipulation concepts
- Développeur - Avancé
  - modélise réel → virtuel
  - produit une interface

## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte
- Concepteur des pièces
  - spécialiste
  - prêtes à être branchées
- Recherche fondamentale
  - énergie
  - matériaux, ...

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...
- Développeur - Intégrateur
  - **composition**
  - **manipulation concepts**
- Développeur - Avancé
  - **modélise réel → virtuel**
  - **produit une interface**



## Automobile

- Conducteur : pilote, taxi ...
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte
- Concepteur des pièces
  - spécialiste
  - prêtes à être branchées
- Recherche fondamentale
  - énergie
  - matériaux, ...

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisir un cloud ...
- Développeur - Intégrateur
  - **composition**
  - **manipulation concepts**
- Développeur - Avancé
  - **modélise réel → virtuel**
  - **produit une interface**
- Recherche fondamentale
  - calculabilité
  - complexité, ...

# Perspective de ce cours



# IP1 résumé

Un programme en Basic, C, Java, Pascal, Php, Python ... est une suite de

- déclarations variables `int x,y;` ( avec int, float, char, boolean )

- opérations mémoire `x=5; y=10; x=3+y;`

- structures de contrôle  
`if (x<y) {x=2*x;} else  
{x=x-1;}`

`while (x<y) {x++ ;}`

`for (int i=0; i < 10; i++ ) {y=y/x;}`

- données plus complexes `int [ ] t1=new int [10]; char [ ][ ] t2;`

- modularité `public static int methodeAdd(int a, int b){return a+b;}`

# (IP1 confort syntaxique - 1)

nuances while / do while :

```
int x=3; // une valeur initiale
do {
    System.out.println("Bonjour");;
    x = x - 1;
} while( x > 0 );
```

Si la valeur initiale de  $x$  est  $> 0$ , affichera  $x$  fois "Bonjour"  
Et si elle est  $\leq 0$ , affichera quand même une fois "Bonjour"

```
int x=3; // une valeur initiale
while( x > 0 ) {
    System.out.println("Bonjour");;
    x = x - 1;
};
```

Idem si la valeur initiale de  $x$  est  $> 0$ , mais si elle est  $\leq 0$ , n'affichera rien

# (IP1 confort syntaxique - 1)

nuances while / do while :

```
int x=3; // une valeur initiale
do {
    System.out.println("Bonjour");
    x = x - 1;
} while( x > 0 );
```

Si la valeur initiale de  $x$  est  $> 0$ , affichera  $x$  fois "Bonjour"  
Et si elle est  $\leq 0$ , affichera quand même une fois "Bonjour"

```
int x=3; // une valeur initiale
if (x<=0) {
    System.out.println("Bonjour"); x = x - 1;
} else {
    while( x > 0 ) {
        System.out.println("Bonjour");
        x = x - 1;
    };
}
```

# (IP1 confort syntaxique - 2)

Syntaxe alternative pour les énumérations de cas :

```
int num = 2;
String jour;
if (num==1) jour = "Lundi";
else if (num==2) jour = "Mardi";
else if (num==3) jour = "Mercredi";
else if (num==4) jour = "Jeudi";
else if (num==5) jour = "Vendredi";
else if (num==6) jour = "Samedi";
else if (num==7) jour = "Dimanche";
else jour = "Numéro Invalide";
```

## (IP1 confort syntaxique - 2)

Syntaxe alternative pour les énumérations de cas :

```
int num = 2;
String jour;
switch (num) {
    case 1: jour = "Lundi"; break;
    case 2: jour = "Mardi"; break;
    case 3: jour = "Mercredi"; break;
    case 4: jour = "Jeudi"; break;
    case 5: jour = "Vendredi"; break;
    case 6: jour = "Samedi"; break;
    case 7: jour = "Dimanche"; break;
    default: jour = "Numéro Invalide";
}
```

## (IP1 "confort syntaxique" - 3)

Syntaxe alternative pour test/affectation rapides :

```
int min;  
if (a<b) min=a;  
else min=b;
```

```
int min = a<b ? a : b;
```



## (IP1 confort syntaxique - 4)

Syntaxe alternative pour parcourir tous les éléments de tableaux :

```
char [] t={'a','b','c'};  
for (int i=0; i<t.length; i++){  
    System.out.println( t[i] );  
}
```

```
char [] t={'a','b','c'};  
for (char val:t){  
    System.out.println( val );  
}
```

# (IP1 confort syntaxique - 5)

Idée :

fournir une méthode qui s'utilise avec un nombre variable d'arguments

```
printByLine("ligne0");  
printByLine("ligne0", "ligne1");  
...  
printByLine("ligne0", "ligne1", "ligne2", "ligne3", "ligne4");  
...
```

Qui donnerait :

ligne0

ligne0

ligne1

etc

# (IP1 confort syntaxique - 5)

Idée :

fournir une méthode qui s'utilise avec un nombre variable d'arguments

```
printByLine("ligne0");  
printByLine("ligne0", "ligne1");  
...  
printByLine("ligne0", "ligne1", "ligne2", "ligne3", "ligne4");  
...
```

Solution limitée : écrire toutes les signatures nécessaires.

```
public static void printByLine(String s1, String s2){  
    System.out.println(s1);  
    System.out.println(s2);  
    System.out.println();  
}
```

## (IP1 confort syntaxique - 5)

Idee :

fournir une méthode qui s'utilise avec un nombre variable d'arguments

```
println("ligne0");  
println("ligne0", "ligne1");  
...  
println("ligne0", "ligne1", "ligne2", "ligne3", "ligne4");  
...
```

Solution limitée : écrire toutes les signatures nécessaires.

```
public static void println(String s1, String s2){code précédent}  
public static void println(String s1, String s2, String s3){  
    System.out.println(s1);  
    System.out.println(s2);  
    System.out.println(s3);  
    System.out.println();  
}
```

etc ... elles peuvent coexister, mais impossible de toutes les écrire

## (IP1 confort syntaxique - 5)

### Idée :

fournir une méthode qui s'utilise avec un nombre variable d'arguments

```
printByLine("ligne0");  
printByLine("ligne0", "ligne1");  
...  
printByLine("ligne0", "ligne1", "ligne2", "ligne3", "ligne4");  
...
```

### Solution possible :

```
public static void printByLine(String ... arg){  
    // "arg" est disponible ensuite sous forme de tableau dans la méthode  
    for(int i=0;i<arg.length;i++) System.out.println(arg[i]);  
    System.out.println();  
}
```

### Nouveauté :

Les ... viennent compléter un type,  
et permettent d'écrire une famille de méthodes

# (IP1 confort syntaxique - 5)

## Idée :

fournir une méthode qui s'utilise avec un nombre variable d'arguments

```
printByLine("ligne0");  
printByLine("ligne0", "ligne1");  
...  
printByLine("ligne0", "ligne1", "ligne2", "ligne3", "ligne4");  
...
```

## Solution possible :

```
public static void printByLine(String ... arg){  
    // "arg" est disponible ensuite sous forme de tableau dans la méthode  
    for(String s:arg) System.out.println(s);  
    System.out.println();  
}
```

## Nouveauté :

Les ... viennent compléter un type,  
et permettent d'écrire une famille de méthodes

# (IP1 confort syntaxique - 5)

```
public static void printByLine(String ... arg){  
    for(String s:arg) System.out.println(s);  
    System.out.println();  
}
```

Est assez comparable à :

```
public static void printByLine(String [] arg){  
    for(String s:arg) System.out.println(s);  
    System.out.println();  
}
```

Mais :

Seule la première remplit le cahier des charges.

Elles ne peuvent coexister.

# (IP1 confort programmation)

Au S1 vous appreniez le langage :

- l'utilisation du couple emacs/javac se justifiait

Vous êtes maintenant encouragés à utiliser **Eclipse** ou **Netbeans** ou **Intellij** ...qui sont des IDE

- environnement de travail moins austère que la console ou emacs
- erreurs de syntaxes détectées à la frappe : gain de temps !
- autre :
  - complétion
  - documentation java
  - organisation de votre travail (package etc)
  - ...

A installer chez vous

Quand ? Ce soir !



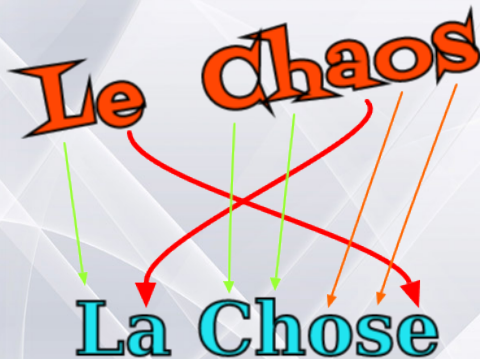
# Analogie ingénieur Automobile/Informatique

## Automobile

- Conducteur
- Bricoleur
  - vidange
  - plaquettes
  - ...
- Garagiste - Constructeur
  - monte / démonte
  - adapte
- Concepteur des pièces
  - spécialiste
  - prêtes à être branchées
- Recherche fondamentale
  - énergie
  - matériaux

## Informatique

- Utilisateur
- Bricoleur
  - installe logiciel
  - imprimante en wifi
  - choisit un cloud ...
- Développeur - Intégrateur
  - composition
  - manipulation concepts
- Développeur - Avancé
  - modélise réel → virtuel
  - produit une interface
- Recherche fondamentale
  - calculabilité
  - complexité



Tout un travail conceptuel consiste à définir des objets

# Les objets encapsulent des données - Exemple d'un cercle

(on ne parle pas de l'héritage ce semestre)

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
}
```

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c;  
    }  
}
```

## Conventions syntaxique

- majuscule au nom de la classe
- la classe est sauvegardée dans un fichier dont le nom est identique à celui de la classe

# Les objets encapsulent des données - Exemple d'un cercle

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
}
```

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c1,c2,c3,c4;  
    }  
}
```

# Les objets encapsulent des données - Exemple d'un cercle

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
}
```

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c1,c2,c3,c4;  
        Cercle [] tab;  
    }  
}
```

# Les objets encapsulent des données - Exemple d'un cercle

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
}
```

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c1,c2,c3,c4;  
        Cercle [] tab;  
    }  
}
```

Ce sont des cercles **différents** (des instances différentes du même type)  
Se pose la question de leur initialisation ...

# Les objets sont des types références (des pointeurs)

comme les tableaux

Leur nature : une référence (une adresse mémoire)

Ils peuvent être initialisés à `null` par exemple. C'est la valeur par défaut

fichier : Test.java

```
public class Test{
    public static void main(String [] args){
        int [] t=null;
        Cercle c=null;
        System.out.println(t); // affiche null
        System.out.println(c); // affiche null
    }
}
```

`null` est une valeur compatible pour tous les types références  
(tableaux ou objets)

# Les objets sont des types références (des pointeurs)

comme les tableaux

Mais ce sont des mondes différents !

fichier : Test.java

```
public class Test{
    public static void main(String [] args){
        int [] t=null;
        Cercle c=null;
        System.out.println(t); // affiche null
        System.out.println(c); // affiche null
        if (c==t) System.out.println("c'est autorisé ça ?") // NON
    }
}
```

A la compilation :

Uncompilable source code - incomparable types : int[] and Cercle



# Les objets sont des types références (des pointeurs)

comme les tableaux

la construction de tableaux s'étend aux tableaux d'objets

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        int [] t = new int [10];  
        Cercle [] tab = new Cercle [10];  
    }  
}
```

# Les objets sont des types références (des pointeurs)

comme les tableaux

La construction de tableaux s'étend aux tableaux d'objets

fichier : Test.java

```
public class Test{
    public static void main(String [] args){
        int [] t = new int [10];
        Cercle [] tab = new Cercle [10];
        for(int val:t) System.out.println(val); // affiche des 0

        for(Cercle c:tab) System.out.println(c) // affiche des null

        // (rappel) la boucle précédente est équivalente à :
        for (int i=0;i<tab.length;i++) System.out.println(tab[i]);
    }
}
```

Mais on n'a pas encore construit un seul cercle !

# Le couple new / constructeur

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c;  
        c = new Cercle(100,200,20); // en position (100,200) et de rayon 20  
    }  
}
```

Le constructeur **avec ces types de paramètres** doit être défini dans la classe Cercle !

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
    ...  
}
```

# Le couple new / constructeur

fichier : Test.java

```
public class Test{
    public static void main(String [] args){
        Cercle c;
        c = new Cercle(100,200,20); // en position (100,200) et de rayon 20
    }
}
```

fichier : Cercle.java

```
public class Cercle{
    int x,y; // coordonnées du centre
    int rayon;
    Cercle (int a,int b, int c){ // même nom que la classe !
        // pas besoin de type retour
        x=a; y=b;
        rayon=c;
    }
}
```

# Le couple new / constructeur

plusieurs constructeurs sont possibles

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c;  
        c = new Cercle(20); // de rayon 20, centré où ?  
    }  
}
```

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
    ...  
}
```

# Le couple new / constructeur

plusieurs constructeurs sont possibles

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Cercle c;  
        c = new Cercle(20); // de rayon 20, centré où ?  
    }  
}
```

fichier : Cercle.java

```
public class Cercle{  
    int x,y; // coordonnées du centre  
    int rayon;  
    Cercle (int d){ // même nom que la classe, pas de type retour  
        x=0; y=0;  
        rayon=d;  
    }  
}
```

# Exercice : modéliser un triangle

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Triangle t;  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    ...  
}
```

# Exercice : modéliser un triangle

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Triangle t;  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    int x1,y1; // coordonnées A  
    int x2,y2; // coordonnées B  
    int x3,y3; // coordonnées C  
    ...  
}
```

Un peu lourd ... heureusement qu'on n'a pas modélisé un carré



# Exercice : modéliser un triangle

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Triangle t;  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    int [] abscisses; // de taille 3  
    int [] ordonnées; // de taille 3  
    ...  
}
```

moins lourd ... mais on peut faire plus clair

# Exercice : modéliser un triangle

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Triangle t;  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    int [] coordonnées; // de taille 6  
    ...  
}
```

encore moins lourd ... mais encore moins clair

# Exercice : modéliser un triangle

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Triangle t;  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    Point a,b,c;  
    ...  
}
```

# Exercice : modéliser un triangle

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Triangle t;  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    Point [] tab;  
    ...  
}
```

# Exercice : modéliser un triangle

fichier : Test.java

fichier : Triangle.java

```
public class Triangle{  
    Point [] tab;  
    ...  
}
```

fichier : Point.java

```
public class Point{  
    int x,y;  
    Point(int a,int b){ // un constructeur  
        x=a; y=b;  
    }  
}
```

# Exercice : modéliser un triangle en plus du cercle

fichier : Test.java

fichier : Triangle.java

```
public class Triangle{  
    Point [] tab;  
    ...  
}
```

fichier : Point.java

```
public class Point{  
    int x,y;  
    Point(int a,int b){  
        x=a; y=b;  
    }  
}
```

fichier : Cercle.java

```
public class Cercle{  
    int x,y;  
    int rayon;  
    Cercle (int a,int b,int c){  
        x=a; y=b;  
        rayon=c;  
    }  
}
```

# Exercice : modéliser un triangle en plus du cercle

fichier : Test.java

fichier : Triangle.java

```
public class Triangle{  
    Point [] tab;  
    ...  
}
```

fichier : Point.java

```
public class Point{  
    int x,y;  
    Point(int a,int b){  
        x=a; y=b;  
    }  
}
```

fichier : Cercle.java

```
public class Cercle{  
    Point p;  
    int rayon;  
    Cercle (int a,int b,int c){  
        p=new Point(a,b);  
        rayon=c;  
    }  
}
```

# Exercice : modéliser un triangle en plus du cercle

fichier : Test.java

fichier : Triangle.java

```
public class Triangle{  
    Point [] tab;  
    ...  
}
```

fichier : Point.java

```
public class Point{  
    int x,y;  
    Point(int a,int b){  
        x=a; y=b;  
    }  
}
```

fichier : Cercle.java

```
public class Cercle{  
    Point p;  
    int rayon;  
    Cercle (int a,int b,int c){  
        p=new Point(a,b);  
        rayon=c;  
    }  
    Cercle (Point x, int d){  
        p=x; // partage de référence  
        rayon=d;  
    }  
}
```



# Exercice : modéliser un triangle en plus du cercle

fichier : Test.java

fichier : Triangle.java

```
public class Triangle{
    Point [] tab;
    Triangle(Point a,Point b,Point c){
        tab=new Point[3];
        tab[0]=a;
        tab[1]=b; tab[2]=c;
    }
}
```

fichier : Point.java

fichier : Cercle.java

```
public class Cercle{
    Point p;
    int rayon;
    Cercle (int a,int b,int c){
        p=new Point(a,b);
        rayon=c;
    }
    Cercle (Point x, int d){
        p=x; // partage de référence
        rayon=d;
    }
}
```

# Objets et niveaux d'abstractions

- Manipuler un objet permet d'améliorer la compréhension globale.

fichier : Test.java

```
Cercle [] tab= new Cercle[2];  
tab[0] = new Cercle (0,0,10);  
tab[1] = new Cercle (1,1,5);  
Cercle tmp=tab[0];  
tab[0]=tab[1];  
tab[1]=tmp;
```

- Un objet (conçu pour regrouper des informations) peut aussi être détaillé/décomposé (on regarde ce qu'il contient)

fichier : Test.java

```
Cercle c= new Cercle(0,0,10);  
int diametre = c.rayon*2; // on pénètre la structure avec le .  
Point centre = c.p;  
int abscisse = c.p.x;  
int ordonnée = centre.y;  
centre.x=-1; // remarquez les changements induits
```

# Classes : définir un modèle et regrouper des méthodes

Le fichier définissant la classe d'objet :

- Porte le nom de la classe
- Contient les attributs/champs définissant les objets
- Contient le/les constructeurs
- Contient les méthodes qui "concernent"<sup>1</sup> ces objets

fichier : Point.java

```
public class Point{
    int x,y;
    Point (int a,int b) {x=a; y=b;}
    public static double distance( Point a , Point b){
        int dx = b.x - a.x;
        int dy = b.y - a.y;
        return Math.sqrt ( dx*dx + dy*dy); // la racine carrée
    }
}
```

1. c'est parfois un peu subjectif

# Classes : définir un modèle et regrouper des méthodes

- Contient les méthodes qui "concernent"<sup>1</sup> ces objets

fichier : Point.java

```
public class Point{
    int x,y;
    Point (int a,int b) {x=a; y=b;}
    public static double distance( Point a , Point b){
        int dx = b.x - a.x;
        int dy = b.y - a.y;
        return Math.sqrt ( dx*dx + dy*dy); // la racine carrée
    }
}
```

- Remarquez l'appel externe à la méthode statique sqrt de Math
- Le même mécanisme permet `Point.distance(arg1,arg2)`

---

1. c'est parfois un peu subjectif

# Classes : définir un modèle et regrouper des méthodes

## Application - Exemple

Si on s'intéresse au périmètre d'un triangle :

- la méthode statique **périmètre** est à définir dans la classe **Triangle**
- elle fait appel à **distance** écrite dans **Point**
- elle même fait appel à **sqrt** écrite dans **Math**

fichier : Triangle.java

```
public class Triangle{
    Point [] tab;
    // ... et ici il y a toujours nos constructeurs ...
    public static double perimetre(Triangle t){ // nouvelle méthode
        double rep= Point.distance( t.tab[0] , t.tab[1] );
        rep += Point.distance( t.tab[1] , t.tab[2] );
        rep += Point.distance( t.tab[2] , t.tab[0] );
        return rep;
    }
}
```

# Petit exercice - Complétez le code

fichier : Triangle.java

```
public class Triangle{  
    // qui contient  
    public static double perimetre(Triangle t){ // etc }  
}
```

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Point a=new Point(0,0);  
        Point b=new Point(10,10);  
        Point c=new Point(20,0);  
        Triangle t=new Triangle(a,b,c);  
        // calcul du périmètre de t  
        double longueur = ... ; // à compléter  
        System.out.println(" Le périmètre est " + longueur);  
    }  
}
```

fichier : Triangle.java

```
public class Triangle{  
    // qui contient  
    public static double perimetre(Triangle t){ // etc }  
}
```

fichier : Test.java

```
public class Test{  
    public static void main(String [] args){  
        Point a=new Point(0,0);  
        Point b=new Point(10,10);  
        Point c=new Point(20,0);  
        Triangle t=new Triangle(a,b,c);  
        // calcul du périmètre de t  
        double longueur = Triangle.perimetre(t);  
        System.out.println(" Le périmètre est " + longueur);  
    }  
}
```

- Remarquez que l'on peut modéliser un cercle par 2 points d'un de ses diamètres. Définissez cette classe en suivant cette remarque, et écrivez une méthode périmètre pour ces cercles.



- Réfléchissez à un environnement où on devrait manipuler à la fois une modélisation d'un permis de conduire à points et d'une police d'assurance. Combien de classes voudriez vous définir ?  
Comment retire t'on des points ? Jusqu'à quand ?  
Peut-on assurer un conducteur qui n'a plus de points à son permis ?