Objets

*** Entité fermée douée de mémoire et de capacité de traitement,**
*** agissant sur réception d'un message,**
*** pouvant fournir un résultat.**

*** Un objet est formé de :**

- données => définissent ce qu'il est,
- programmes ou procédures => définissent ce qu'il peut faire,

Classes

*** Créer un objet générique "MOULE ou PROTOTYPE",**

*** CLASSE => modèle décrivant le contenu et le comportement des futurs objets de la classe, ensemble d'objets,**

- le contenu = les données,

- le comportement = les méthodes,

*** INSTANCIATION**

fabrication à partir du modèle de la classe, d'un objet particulier, élément de cette classe.

Public & Private

Les variables membres sont en général private.

Les méthodes, en fonction du besoin.

public int x ;	**private float z ;**
public void affiche() ;	**private int genere(int) ;**

Classes, champs et méthodes

```
public class compte {

   // Définition des champs
   String nom;
   double actif;
   int limite; // limite de crédit
```



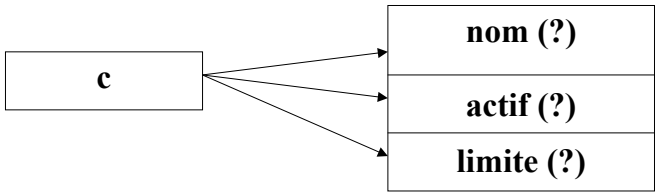
```
   // Définition des méthodes

   void initialise(){
      nom = "";
      actif = 0.0;
      limite = 2000;
   }
   double depot(double argent){
      actif += argent;
      return actif;
   }
```



```
   void affiche() {
      System.out.println("nom\
            de la personne: " + nom);
      System.out.println("actif\
            de son compte: " + actif);
      System.out.println("limite\
               de crédit: " + limite);
   }
}
```


déclaration :

```
compte c; // pas d'espace mémoire réservé.
```



un emplacement
pour une référence
à un objet de type compte.

création de l'objet :

```
compte c = new compte();
```

création d'un objet de type compte et place sa référence dans c.

```
c.initialise();
c.depot(50);
c.affiche();
```


Fichier source et classes

une seule classe dans un fichier source (*.java) ou plusieurs classes?

- plusieurs classes, mais une seule doit être publique.

- la classe contenant la méthode main doit être publique pour que la machine virtuelle puisse y accéder.

Constructeur

Définitions

- initialise dans compte permet d'initialiser les champs d'un objet.

- méthode appelée à la création de l'objet, sert à initialiser l'objet

- constructeur permet donc d'automatiser le mécanisme d'initialisation d'un objet.

si la fonction `initialise()` n'a pas été définie dans l'exemple `compte`?

en java, le compilateur initialise les membres données de la classe avec les valeurs de défaut suivantes :

Type du champ	Valeur par défaut
boolean	false
char	caractère de code nul
entier (byte, short, long, int)	0
flottant (float, double)	0.f ou 0.
objet	null

- ne retourne rien (ne pas mettre void)

- public

- porte le même nom que la classe

syntaxe: `nom_de_la_classe(arguments);`

les arguments ne sont pas obligatoires.


```
public class compte {

   // Définition des champs
   String nom;
   double actif;
   int limite; // limite de crédit

   compte(String monnom,\
        double monactif,int malimite){
     nom = monnom ;
     actif = monactif ;
     limite = malimite ;
   }
```


- Constructeur par défaut

Chaque classe a un constructeur par défaut (il est caché, mais il existe),

```
compte CB = new compte();
```

appel le constructeur par défaut défini par le compilateur comme étant:

```
compte(){ // fait quelque chose ... }
```

il est appelé si aucun constructeur n'a été défini explicitement dans la classe,

si, dans une classe, il a été défini un constructeur (comme vu précédemment dans le cas de la classe `compte`), ce dernier masque le constructeur par défaut.

Ainsi, le constructeur par défaut cesse d'exister et devient donc inaccessible ; de ce fait, l'instruction suivante :

```
compte CB new compte();
```

génère une erreur, car il n'existe aucun constructeur

- Un constructeur ne peut pas être appelé directement depuis une autre méthode :

compte c = new compte() ;

c.compte() ;

- Un constructeur peut appeler un autre constructeur de la même classe :

compte (`String,double,int)=>` compte

- public vs. private.

Construction et initialisation des objets

- initialisation par défaut (déjà vu).

- initialisation explicite lors de la déclaration des champs.

- exécution des instructions du corps du constructeur.

class A {
public A(…) {// etc.}
int n=10;
int p;
}
	A a = new A(…);

class A {
public A(…) {// etc.}
int n=10;
int p = n+5;
}
	A a = new A(…);

Attention à l'ordre :

```
int p = n+5;
int n=10;
```


attribut final et constructeur

```
class A {
    public final int x =10;
}
```

class A {
 	
public A() {
x = 10;
}
public final int x;
}
	class A {
	
	public A(int nn) {
	x = nn;
	}
	public final int x;
	}

- attribut final => initialisé qu'une seule fois.

- champ déclaré final doit être initialisé au plus tard dans le constructeur.

- ne contez pas trop sur l'initialisation par défaut pour faire le travail …

```
class A {
    public A() {// on ne touche pas à x.}
    public final int x; // erreur de compilation
}
```

=> de préférence oublier l'existence même de l'initialisation par défaut.

Assesseurs et modificateurs

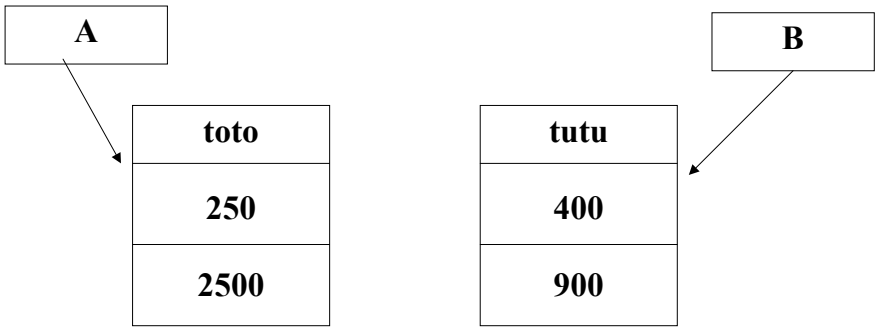
Assesseurs (accessor) : les méthodes get. Donnes des informations relatives aux valeurs de certains champs.

Modificateurs (mutator) : les méthodes set. Modifies les valeurs de certains champs (état d'un objet).

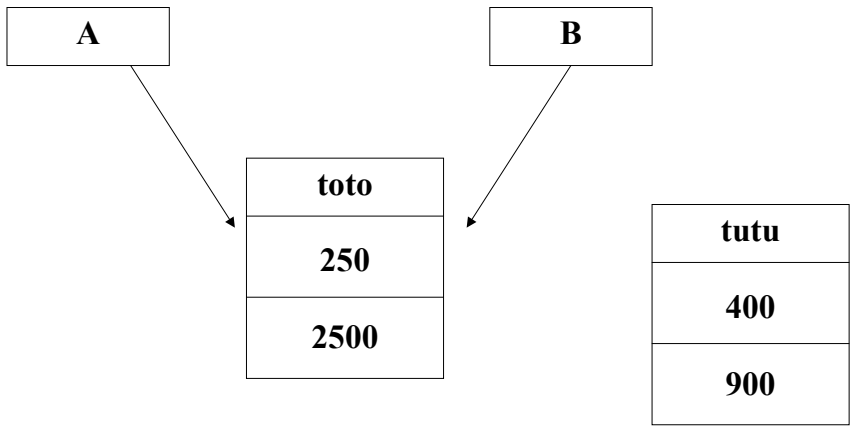
Méthodes de services : généralement privées.

Affectation et comparaison

```
compte A = new compte("toto",250.,2500);
compte B = new compte("tutu",400.,900);
```



A=B;



```
compte A = new compte("toto",250.,2500);
compte B = A.copie();

class compte {

    public compte copie() {
        compte X = new compte(nom,actif,limite);
        return X;
    }
}
```


sinon la notion de clonage (clone) en java, on reviendra dessus.

A!=B; // on compare les références et non pas les objets.

if (A.equals(B)) {} // retourne true si c'est le cas.

copie superficielle et copie profonde

Une copie superficielle recopie la valeur de tous les champs y compris ceux de type classe.

Une copie profonde recopie la valeur de tous les champs et pour les champs de type classe, elle crée une nouvelle référence à un

objet du même type et ayant les mêmes valeurs, mais l'emplacement mémoire (la référence) n'est pas le même.

Destructeur

Appelé automatiquement avant la destruction de l'objet.

Sert à rendre des ressources aux systèmes prises par l'objet à détruire.

Opération prise en charge par le garbage collector (GC) (ramasse-miettes).

(new compte("toto",250.,2500)).affiche();
nettoyage après affichage => garbage collector!

Méthodes

- peut fournir un résultat sinon mettre void.

objet.methode (liste d'arguments);

(voir les différents exemples pour les notions de variables locales etc.).

Membres statiques

Sont partagés de toutes les instances (objets) d'une classe.

Ils sont accessibles indépendamment de l'existence de l'instance d'une classe.

Accès usuel et recommandé :

Classe.membre

Classe : nom de la classe, membre : nom du membre.

Méthodes statiques

Sont partagées de toutes les instances (objets) d'une classe.

Ils sont accessibles indépendamment de l'existence de l'instance d'une classe.

Accès usuel et recommandé :

Classe.methode
Classe : nom de la classe, membre : nom de la méthode.

Attention : Une méthode statique ne peut accéder qu'à des variables statiques.

Accès aux membres

depuis l'extérieur de la classe …

Impossible si le membre est privé.

Si le membre est static :
de préférence => Classe.membre

Cas classique :

objet.membre (objet est le nom de votre objet).

Autoréférence THIS

`this` dans une fonction membre donne l'adresse de l'objet qui fait l'appel (ou objet receveur).

- utilisable par les méthodes non statiques.

- utile pour faire référence à un objet dans sa globalité et non pas chacun de ses champs.

- permet des appels en cascade de fonctions membres.

- si utilisé dans un constructeur, il doit être obligatoirement la première instruction de ce constructeur.

```
public compte(String nom,double actif,int limite){
    this.nom = nom ;
    this.actif = actif ;
    this.limite = limite ;
}
```