Programmation Objet 1

16h CM, 26h TD, 18h TP

Objectifs : maitriser les mécanismes de base de la programmation objet
apprendre à développer des interfaces graphiques

Prérequis : algorithmique de base
maîtrise du C

Bibliographie : https://docs.oracle.com/javase/tutorial/
https://www.w3schools.com/java/

Contributeurs : Frédéric Fürst
Jean-Luc Guérin

Java est un langage de programmation généraliste, orienté objet, fortement typé.

Java a été initialement développé par Sun Microsystems (v1.0 en 1995), racheté en 2009 par Oracle.

OpenJDK est une implémentation libre de Java

Java est un des principaux langages de développement logiciel et offre un écosystème logiciel très riche :

- interfaces graphiques (AWT, SWING)
- interfaçage avec les BD (JDBC, …)
- applications Web coté client (Applets) et coté serveur (Servlet, JSP, …)
- Java 3D

Java est un langage semi-interprété, compilé en byte-code, qui s'exécute sur une machine virtuelle :

→ pas d'édition de lien à la compilation (programmes plus légers)
→ grande portabilité des programmes entre systèmes
→ exécution parfois plus lente

Le compilateur est installé avec le JDK (Java Development Kit)

La machine virtuelle est généralement déjà installée dans les OS avec le JRE (Java Runtime Environment)

Java sur le site d'Oracle : https://www.oracle.com/fr/java/technologies/

Documentation de l'API Java : https://docs.oracle.com/en/java/javase/17/

Oracle recommande fortement l'utilisation d'une version LTS (long-term support) de JAVA (8, 11, 17)

Java n'a pas de pointeurs, la mémoire est gérée automatiquement par un mécanisme de ramasse-miettes (garbage collector) :

- Identifie les objets non utilisés
- Libère l'espace mémoire et le re-compacte éventuellement

Les types de base de Java sont proches de ceux du C :

Type :	`byte`	`short`	`int`	`long`	`float`	`double`	`boolean`	`char`
Taille:	1	2	4	8	4	8	1	2

Java offre un type `String` pour les chaines, et de nombreux autres types d'objets.

Les opérateurs et structures de contrôle sont les mêmes en C et en Java

Le mode de passage des arguments en Java est imposé :

- passage par valeur pour les types primitifs
- passage par référence pour les tableaux et les objets

On a toujours accès à la taille d'un tableau avec l'opérateur `length`

```
int[] tab1 = {1, 5, 8};
char[] tab2 = new char[11];
...
for(int i = 0; i < tab1.length; i++){
    tab1[i] = 1;
}
```

Entrées-sorties en Java :

```
import java.util.Scanner;

System.out.println("Veuillez saisir un mot :");
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
System.out.println("Saisissez un nombre :");
int i = sc.nextInt();
```

Méthode principale en Java :

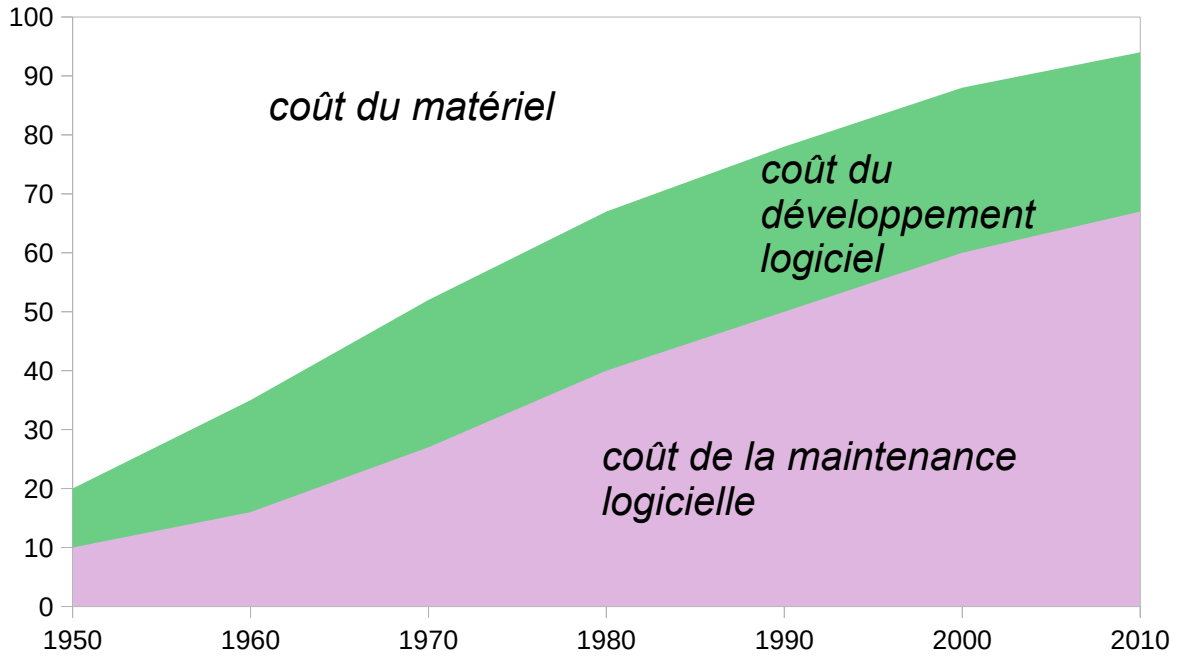
- elle doit être placée dans une classe (voir plus loin)
- le tableau de chaines en argument contient les paramètres tapés en ligne de commande

```
class Truc{

    public static void main(String[] tab){
        if(tab.length>0){
            System.out.println("Le premier paramètre est " + tab[0]);

        }
    }

}
```

```
D:\>cd Boulot\Cours\POO_2018\Cours

D:\Boulot\Cours\POO_2018\Cours>javac Truc.java

D:\Boulot\Cours\POO_2018\Cours>java Truc

D:\Boulot\Cours\POO_2018\Cours>java Truc Youpi
Le premier paramètre est Youpi

D:\Boulot\Cours\POO_2018\Cours>
```

L'informatique décolle dans les années 1970 :

- nouvelles technologies (mode multi-utilisateurs, interfaces graphiques, programmation concurrente, …)
- les programmes grossissent (un gros logiciel fait 10 000 lignes de code en 1970, 50000 en 1980, des millions de nos jours)
- les ordinateurs se banalisent dans le monde du travail, et vont devenir des objets de consommation



De bonnes techniques de développement logiciel deviennent nécessaires.

Utilité : le logiciel doit correspondre aux besoins des utilisateurs

Utilisabilité : ergonomie et facilité d'apprentissage et d'utilisation

Fiabilité : correction et conformité, robustesse et sureté

Efficacité : temps d'exécution faible et utilisation réduite des ressources

Interopérabilité : interactions possibles entre logiciels

Portabilité : le logiciel doit pouvoir tourner sur le plus possible de systèmes

Maintenabilité : facilité de test, concision et lisibilité, extensibilité

Réutilisabilité : le code doit pouvoir être réutilisé au maximum

Généralisation : regrouper du code commun à plusieurs programmes dans un seul programme et regrouper un ensemble de fonctionnalités semblables en une fonctionnalité paramétrable.

Abstraction : décrire des entités à différents niveaux d'abstraction, ne donner que les éléments pertinents et omettre ceux qui ne le sont pas.

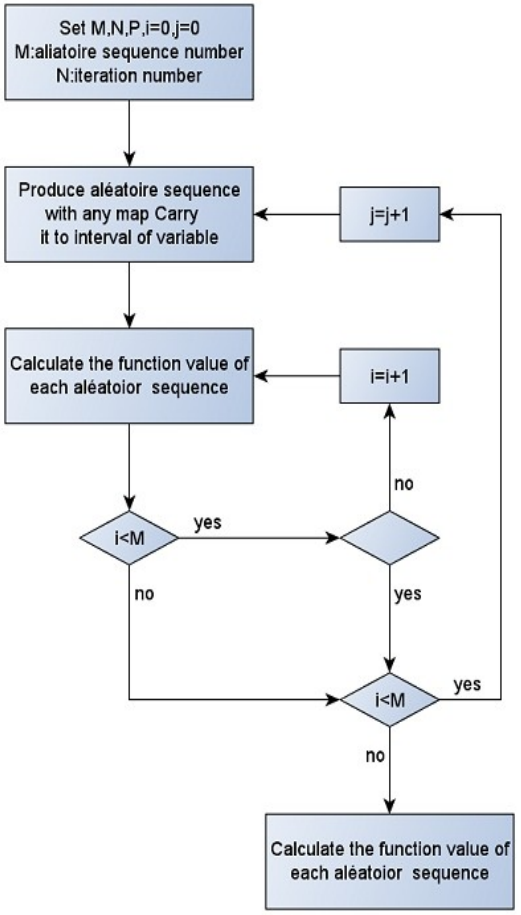
Modularité : décomposer les logiciels en composants aussi petits et indépendants que possible.

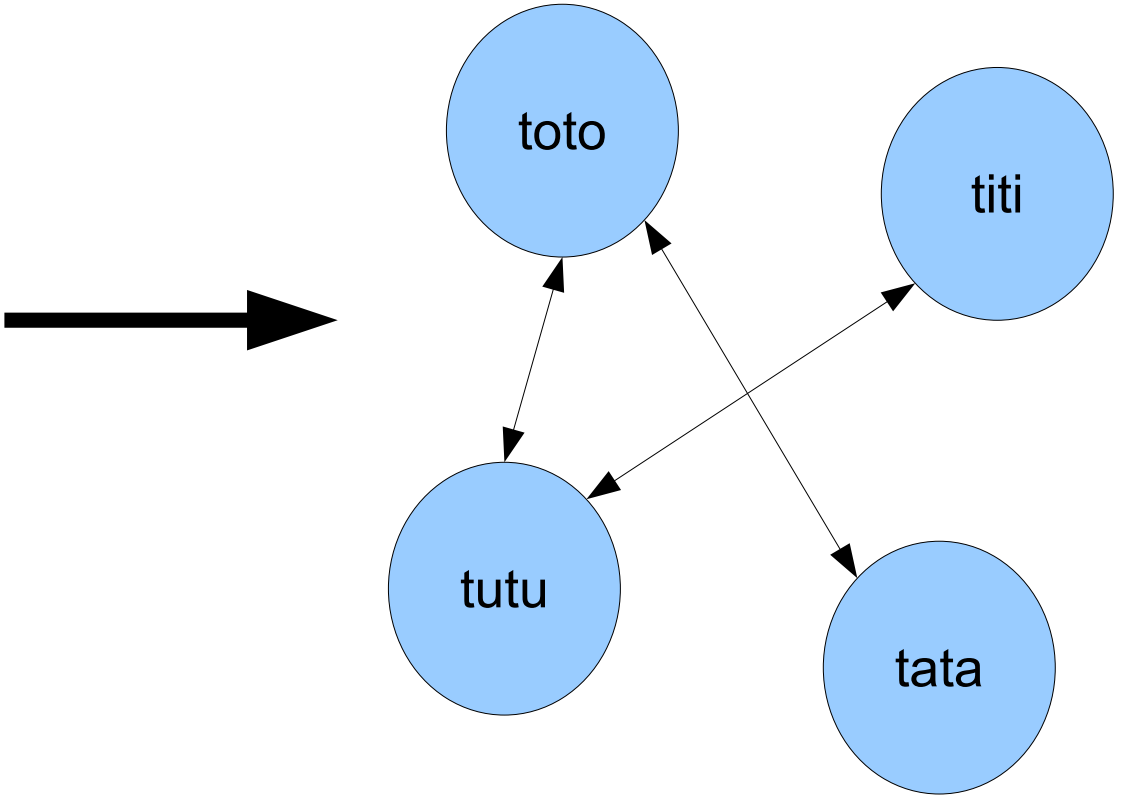
Documentation : documenter le code et produire des notices permettant la réutilisation.

Vérification : respecter des spécifications initiales, permettre des tests unitaires.

La programmation objet est inventée dans les années 1970 comme une extension de la programmation procédurale.

L'idée est de concevoir les programmes non plus comme des lignes de codes qui s'exécutent séquentiellement, mais comme des objets qui dialoguent.





Simula 64 est le premier langage à utiliser ce paradigme mais de façon limitée.

Smalltalk, développé à partir de 1971 par **Alan Kay** et publié en 1980, est le premier vrai langage objet.

D'autres suivront : Eiffel en 1986, CLOS en 1988, Python en 1991, ...

Des versions objets de langages existants seront développées : C++ et Objective-C pour le C, ADA95 pour le ADA, Object Pascal pour le Pascal, PHP5 pour le PHP, ...

Première version de Java en 1995, développée principalement par **James Gosling**.

La programmation objet devient à la fin des années 90 le paradigme dominant en programmation.

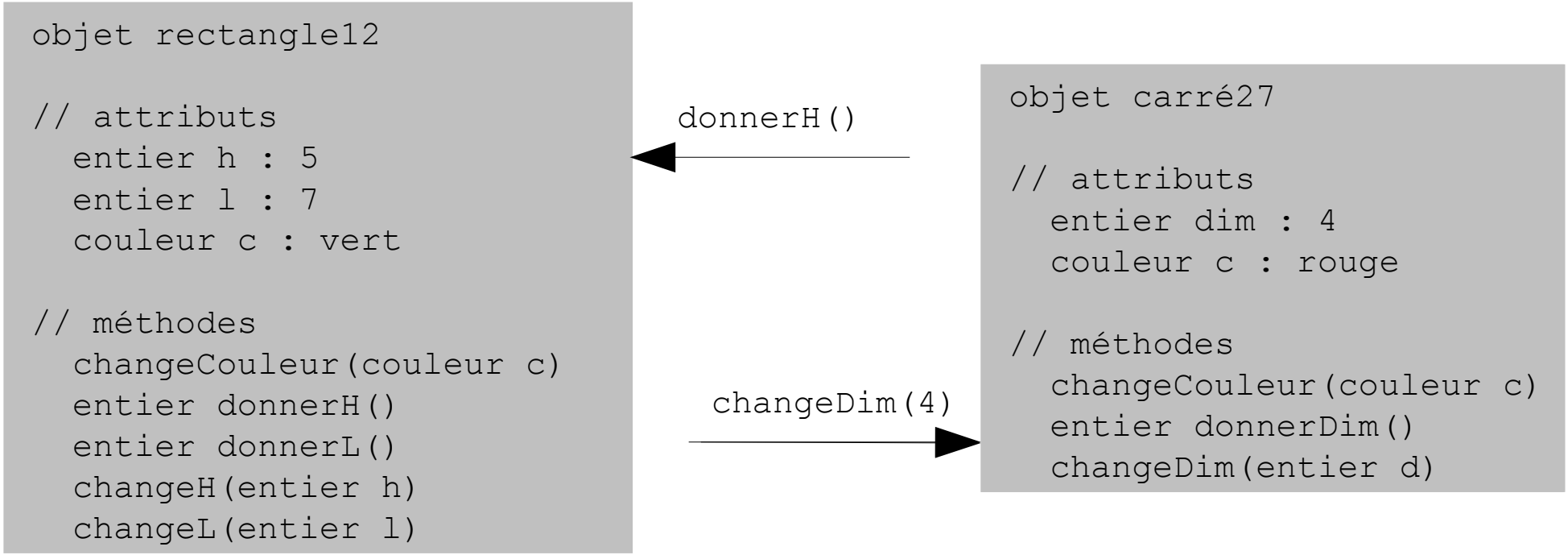
Les technologies objets englobent désormais :

- la programmation (langages objets, composants, ...)
- la modélisation logicielle (UML, …)
- la communication entre programmes (CORBA, …)
- les ateliers logiciels (Eclipse, NetBeans, …)
- les règles de conception (Design Pattern, ...).

Chaque objet en mémoire regroupe :

- des attributs : valeurs typées qui décrivent les caractéristiques de l'objet
- des méthodes : fonctions qui décrivent ce que l'objet peut faire et qui opèrent sur les attributs des objets

Les messages que s'échangent les objets sont des appels de fonction.



L'envoi d'un message s'effectue en préfixant le message par la variable contenant l'objet auquel il est destiné.

Exemple : `r.changeCouleur(bleu)` si `r` contient l'objet `rectangle12`.

Le programme est lancé par l'appel d'une fonction principale (main) qui peut être placée dans le code de n'importe quel objet.

Pour lancer le programme, il faut préciser quelle fonction principale est exécutée.

```
objet rectangle12

  ...
  // z doit contenir carré27
  z.changeDim(4);
  ...
  main(){
    ...
  }
```

```
objet carré27

  ...
  main(){
    ...
    // v doit contenir
 rectangle12
    entier e = v.donnerH();
  }
  ...
```

Les attributs des objets peuvent être des objets également.

Les paramètres des méthodes et leurs valeurs de retour peuvent également être des objets.

```
objet facteur12527

// attributs
  chaine nom : "Olivier"
  entier matricule : 12527
  Centre_de_Tri ct : ct234

// méthodes
  distribuerCourrier()
  trierCourrier()
```

```
objet ct234

// attributs
  chaine nom : "Centre de Tri de Pétaouchnok"
  Facteur[] personnel : tab456

// méthodes
  payerTousLesFacteurs()
  embaucher(Facteur f)
  virer(Facteur f)
```

Idéalement, tout est objet (en Smalltalk par exemple).

En Java, il existe des types primitifs non objets.

Un enregistrement est un objet sans méthode.

Les enregistrements sont décrits par des types enregistrement, qui spécifient les noms des champs et leurs types.

```
struct Personne{
    char* nom;
    char* prenom;
    int age;
}
```

```
enregistrement e1
    nom : "toto"
    prenom : "tutu"
    age : 27
```

```
enregistrement e2
    nom : "toto"
    prenom : "titi"
    age : 32
```

```
enregistrement e3
    nom : "tata"
    prenom : "tete"
    age : 12
```

Les objets sont décrits par des classes, qui spécifient les noms des champs (attributs) et leurs types, mais aussi les méthodes des objets.

Plusieurs raisons justifient l'existence des classes :

Typage : typer les objets permet de tester à la compilation la correction des programmes. Les classes sont les types des objets.

Généralisation : des objets peuvent avoir des contenus semblables susceptibles d'être décrits de la même façon

- même nombre d'attributs, avec les mêmes types
- mêmes méthodes

Il y a tout intérêt à regrouper les méthodes et la déclaration des types des attributs dans une même entité : c'est une classe.

La plupart des langages objets permettent donc d'écrire des classes.

Une classe définit :

- des attributs avec leurs noms et leurs types. Les valeurs des attributs ne sont a priori pas définis dans la classe, car propres à chaque objet. Une valeur par défaut peut cependant être spécifiée.

- des méthodes avec leur signature et le code qui décrit le comportement associé.

```
class Personne{
    String nom;
    String prenom;
    int age = 0;

    public String donneNom(){
        return nom;
    }

    public void anniversaire(){
        age = age + 1;
    }

    public void changePrenom(String s){
        prenom = s;
    }
}
```

Les attributs et les méthodes sont appelés membres de la classe. Le membre `m` d'un objet `o` est désigné par `o.m`

Les objets décrits par une classe `C` sont des instances de la classe `C`

```
class Multiplieur{
   int multiplicande;

   int multiplie(int e){
      return multiplicande*e;
   }

}
```

```
objet tripleur instance de Multiplieur
  multiplicande : 3

  int multiplie(int e){
    return multiplicande*e;
  }
```

```
objet doubleur instance de Multiplieur
  multiplicande : 2

  int multiplie(int e){
    return multiplicande*e;
  }
```

```
Multiplieur m;
...
int i = m.multiplie(5);
```

Quelle valeur sera stockée dans la variable `i` ?

Chaque objet est différent des autres, même si deux objets de même type peuvent avoir les mêmes valeurs d'attributs.

```
objet o1 instance de Personne

  // attributs
  nom : "toto"
  prenom : "titi"
  age : 56

  ...
```

```
objet o2 instance de Personne

  // attributs
  nom : "toto"
  prenom : "titi"
  age : 56

  ...
```

Les classes sont définies par le programmeur dans son programme, les objets sont créés lors de l'exécution du programme.

Les objets peuvent être créés à partir des classes (opérateur `new` en Java) ou autrement si le langage n'implémente pas les classes.

Langages objets à base de classes : chaque objet appartient à (au moins) une classe qui décrit toutes ses instances.

Exemples : Simula, Smalltalk, Java, C++, …

Langages objets à base d'objets (ou de prototypes) : pas de classe, mais un objet peut déléguer ses attributs à ses clônes. Ces langages ne sont généralement pas typés.

Exemple : JavaScript
Depuis la version 6 d'ECMAScript datant de 2015 les classes ont été ajoutées à JavaScript.

Pour un même type de langage, les implémentations du paradigme peuvent être très différentes : Smalltalk et Java par exemple.

Les méthodes sont des fonctions attachées à chaque objet et possèdent :

- un nom.
- des paramètres. En Java, les paramètres sont passés par référence, sauf pour les types primitifs.
- un type de retour (**`void`** si la méthode ne retourne rien).

```
class Facteur{
  ...
  void distribuerCourrier(Courrier c){
    if(c.lePays() == "France"){
      c.affecter(centreDeTriNational);
    }
    else c.affecter(centreDeTriInternational);
  }
}
```

```
class Courrier{
  Pays pays
  Ville ville
  ...
  Pays lePays(){
    return pays;
  }
  void affecter(CentreDeTri ct){
    ct.accepter(this);
  }
}
```

Le mot clé `this` désigne l'objet dans lequel le code est exécuté, c'est à dire l'objet receveur du message. Il peut être utilisé pour désigner un attribut ou une méthode.

Le mot clé `null` est une référence vide (sur aucun objet).

```
class Multiplieur {
  Int multiplicande;

  int multiplie(int e){
    return this.multiplicande*e;
  }

  int multiplie(int e, Multiplieur m){
    return m.multiplie(e);
  }
}
```

```
objet doubleur instance de Multiplieur
  Multiplicande : 2

objet tripleur instance de Multiplieur
  multiplicande : 3
```

Quelles valeurs seront stockées dans `i` et `j` ?

```
Multiplieur x,y;
...
// on met l'objet doubleur dans x
// on met l'objet tripleur dans y
...
int i = x.multiplie(5);
int j = x.multiplie(5,y);
```

En cas d'ambiguité sur la désignation d'un membre, le membre considéré est celui défini le plus localement.

```
class Truc{
    int a;
    int b;

  int m1(int a){
    return this.a*b;
  }

  int m2(int a, Truc t){
    return this.m1(t.a*a);
  }
}
```

```
objet titi instance de Truc
  a : 2
  b : 3

objet toto instance de Truc
  a : 5
  b : 6
```

```
titi.m2(5,toto)?
```

L'envoi d'un message est synchrone : il bloque l'exécution de la méthode appelante jusqu'à ce que la méthode appelée soit terminée, même si la méthode appelée est dans un autre objet.

```
objet o1

  ...
  void methode1(...){
    int i = o2.methode2();
    System.out.println(i);
  }
  ...
```

```
objet o2

  ...
  int methode2(){
    return this.dimension;
  }
  ...
```

La programmation objet est cependant compatible avec la programmation concurrente (multithread).

Dans un programme objet multithread, chaque thread exécute ses propres objets. Chaque objet peut même s'exécuter dans un thread différent.

```
objet facteur12527
  ...
  // méthodes
   void distribuerCourrier(Courrier c){
    if(c.retournePays() == "France")
      c.affecter(centreDeTriNational);
    else
      c.affecter(centreDeTriInternational);
  }
  ...
```

facteur12527.distribuerCourrier (courrier6543)

courrier6543.retournePays()

"Pérou"

courrier6543.affecter(centreDeTriInternational)

```
objet courrier6543
  // attributs
  pays : "Pérou"
  ...
  // méthodes
  Pays retournePays(){
    return pays;
  }
  void affecter(CentreDeTri ct){
    ct.accepter(this);
  }
```

centreDeTriInternational.accepter (courrier6543)

TRUE

```
objet centreDeTriInternational
  // attributs
  adresse : "12 rue Machin ..."
  listeCourriers : (courrier2,...)
  // méthodes
  boolean accepter(Courrier c){
    ...
  }
```

Différentes sortes de méthodes sont distinguées :

- constructeur : réserve une zone mémoire et initialise les valeurs des attributs de l'objet.
- destructeur : libère la zone mémoire occupée par l'objet. Les destructeurs n'existent pas en Java.
- modifieur : modifie la valeur d'un attribut de l'objet.
- accesseur : renvoie la valeur d'un attribut de l'objet.

Pour distinguer entre méthodes, classes, etc, la plupart des langages dont Java utilisent la convention suivante : les noms de classe (types) commencent par une majuscule, tous les autres identifiants commencent par une minuscule.

En Java, un constructeur a le nom de sa classe et aucun type de retour (pas même `void`).

Par convention, un accesseur a pour nom `getA` où `A` est le nom de l'attribut accédé. On parle également de getter pour les accesseurs.

Par convention, un modifieur a pour nom `setA` où `A` est le nom de l'attribut modifié. On parle également de setter pour les modifieurs.

```
classe Facteur{

  // attributs
  String nom;
  int matricule;

  // méthodes
  Facteur(String n, int m){
    nom = n;
    matricule = m;
  }

  String getNom(){
    return nom;
  }

  int getMatricule(){
    return matricule;
  }

  void setNom(String newNom){
    nom = newNom;
  }
}
```

En Java, un objet est créé par appel à un constructeur de sa classe :

L'opérateur `new` déclenche l'allocation mémoire de l'objet et retourne l'instance créée.

```
Facteur f = new Facteur("Olivier", 03984);
```

Si aucun constructeur n'a été défini dans la classe un constructeur par défaut sans paramètre est ajouté par le compilateur.

```
Facteur f = new Facteur();
```

Plusieurs constructeurs peuvent être définis avec des paramètres différents (surcharge).

Les attributs pour lesquels des valeurs par défaut sont spécifiées peuvent ne pas être initialisés dans le constructeur.

```
class Facteur{
  // attributs
  String nom;
  int matricule;
  int age = 0;
  ...
  // méthodes
  Facteur(String n, int m){
    nom = n;
    matricule = m;
  }
  Facteur(String n, int m, int age a){
    nom = n;
    matricule = m;
    age = a;
  }

  ...
}
```

La destruction des objets “inutiles” est réalisée par le mécanisme automatique de ramasse miettes (garbage collector) qui peut être appelé avec la méthode `gc()`.

```
System.gc()
```

Un objet qui n'est plus référencé, n’est plus utilisable et doit disparaître de la mémoire.

Dans d’autres langages, la destructions des objets et Laissée à la charge des développeurs.
C’est le cas de C++ prévoit des méthodes de destruction (désallocation mémoire). Un destructeur porte le nom de la classe et n'a ni paramètre ni type de retour.

```
class Facteur{
  ...
  ~Facteur(){
    ...
  }
}
```

En Java la méthode `finalize` est appelée avant la destruction des objets. Par défaut, elle ne fait rien.

```
class Facteur{
  ...
  protected void finalize(){
    // supprimer l'adresse du facteur dans l'annuaire
    // exécuter les dispositions testamentaires du facteur
  }
}
```

Le modèle objet permet d'écrire un programme sous forme d'un ensemble de classes relativement indépendantes, ce qui augmente la modularité.

Chaque classe peut être écrite par un programmeur différent, ce qui facilite le développement logiciel.

```
programme Toto{
  // déclarations de types
  type Facteur ...
  type Courrier …
  ...

  // déclarations de fonctions
  int f1(...) ...
  void f2(...) ...
  ...

  // programme principal
  public static void main(...)
  ...
}
```

```
class Facteur{
  ...
  int f1(...){
  ...
}
```

```
class Courrier{
  ...
  void f2(...){
  ...
}
```

```
class ...{
  ...
}
```

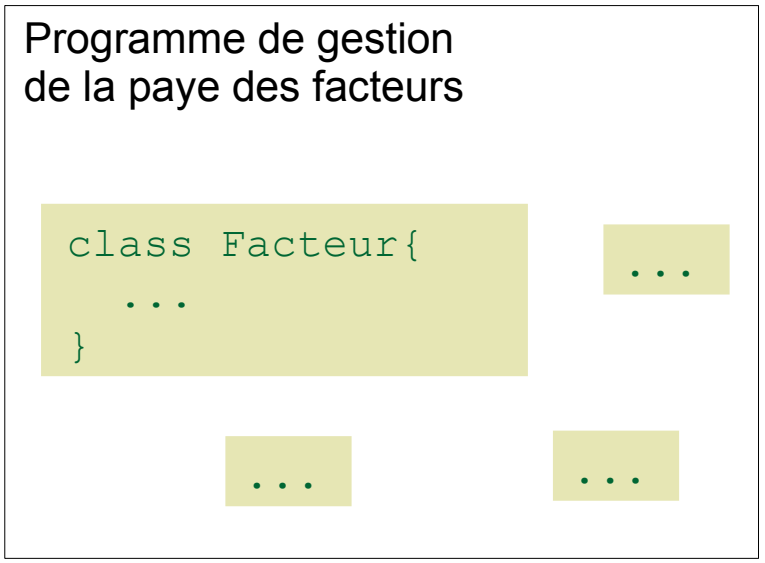
```
...
```

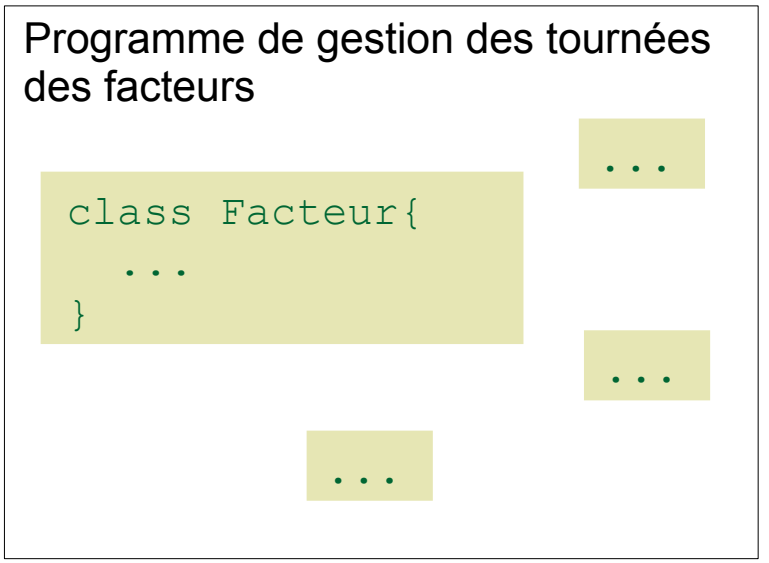
```
...
```

Vérification : un programme objet peut être testé classe par classe

Maintenabilité : modifier une classe peut a priori se faire sans modifier d'autres classes et en ne recompilant que cette classe

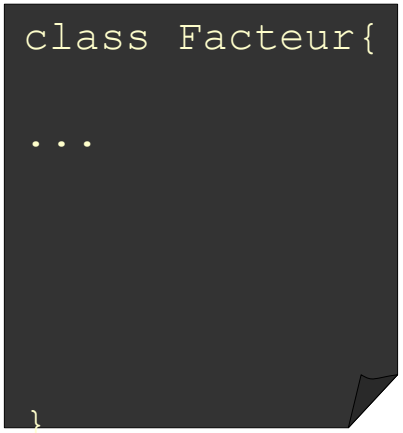
Réutilisabilité : une classe peut facilement être réutilisée partout où l'entité qu'elle représente doit être manipulée.





Généralement, on écrit un fichier par classe pour augmenter la modularité, la maintenabilité et la réutilisabilité des programmes.

En Java, pour les classes publiques (voir plus loin), le nom du fichier doit être identique à celui de la classe.



Facteur.java

Un programme objet va être constitué de plusieurs (de centaines) de classes, dont les fichiers sont organisés en répertoires.

La programmation objet est une approche indépendante des langages de programmation

- certains langages ont été développés directement autour de ce paradigme (Simula, Smalltalk, Eiffel, Java, ...).
- les langages non objet peuvent être étendus avec des mécanismes objet (C, Pascal, PHP, LISP, CAML, ADA, Cobol, Perl, Prolog, ...).

Il s'agit d'un paradigme, pas toujours implémenté de façon contraignante

- un programme écrit dans un langage non objet peut respecter le paradigme objet.
- un programme écrit dans un langage objet peut ne pas respecter ce paradigme.

Programmer (proprement) objet dépend en bonne partie du programmeur!