Opérations UML et leur implémentation par des méthodes Java

LIRMM / Université de Montpellier

Janvier 2017

Classes, opérations et méthodes

Partie précédente

- Définition de classes, d'attributs (partie structurelle)
- Définition de ce qu'est un objet, pas de ce qu'il fait

Méthodes et opérations

- Définissent des comportements des instances de la classe.
- Ex. Pour une classe voiture, exprimer ce que peut faire une voiture : klaxonner, fournir une assistance au parking, etc.
- Peuvent manipuler les attributs, ou faire appel à d'autres méthodes de la classe.
- Peuvent être paramétrées et retourner des résultats.
- Permettent la communication des instances par envoi de messages.

Sommaire

Opérations

- Les opérations sont des éléments du diagramme de classes représentant la dynamique du système

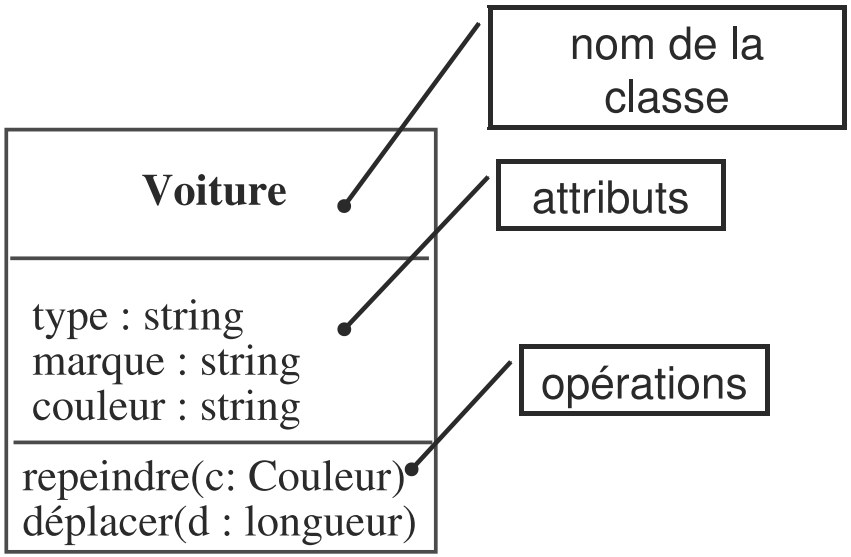


Figure : Les opérations dans les classes UML

Syntaxe

`[visibilité] nom ( lst-paramètres ) [: typeRetour]`
`[lst-propriétés]`
où la syntaxe de chaque paramètre est :
`[direction] nom : type[[multiplicité]] [= valeurParDéfaut]`
`[liste-propriétés]`
avec `direction` $\in \{in, out, inout\}$, et `multiplicité` définit une valeur (1, 2, n, ...) ou une plage de valeurs (1..*, 1..6, ...).

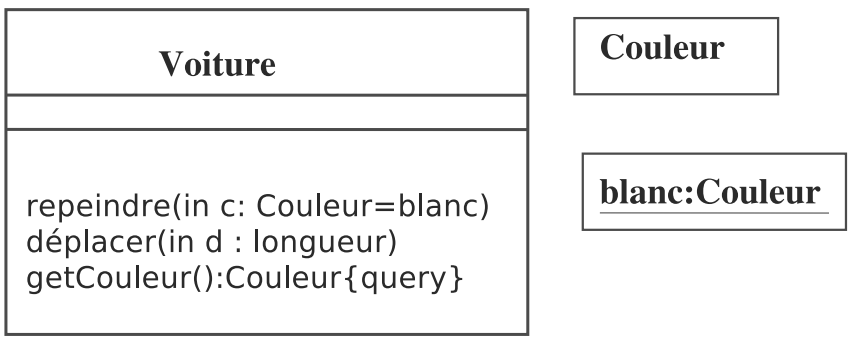


Figure : Exemple d'opérations en UML

Le nom

- Une opération a un nom
- Donner un nom portant le plus de sémantique possible
- Ex. "klaxonner", "déplacer", "repeindre", plutôt que : "o1", "o2", "op"

Visibilité

Dans les grandes lignes :

- Publique. Dénoté +. Signifie que cette opération pourra être appelée par n'importe objet
- Privée. Dénoté -. Signifie que cette opération ne pourra être appelée que par des objets instances de la même classe
- Paquetage. Dénoté $\sim$. Signifie que cette opération ne pourra être appelée que par des objets instances de classes du même paquetage.
- Protégée. Dénoté #. Signifie que cette opération ne pourra être appelée que par des objets instances de la même classe ou d'une de ses sous-classes (on verra plus tard ce que cela signifie exactement)

Paramètres

Modes de passage

in le paramètre est une entrée de l'opération, et pas une sortie : il n'est pas modifié par l'opération. C'est le cas le plus courant. C'est aussi le cas par défaut en UML.

out le paramètre est une sortie de l'opération, et pas une entrée. C'est utile quand on souhaite retourner plusieurs résultats : comme il n'y a qu'un type de retour, on donne les autres résultats dans des paramètres out.

inout le paramètre est à la fois entrée et sortie.

Propriétés

- Propriétés facultatives précisant le type d'opération
- Exemple. `{query}` : l'opération n'a pas d'effet de bord
- Propriétés entre accolades

Opérations de classe

- C'est une opération qui ne s'applique pas à une instance de la classe
- Elle peut être appelée sans avoir instancié la classe.

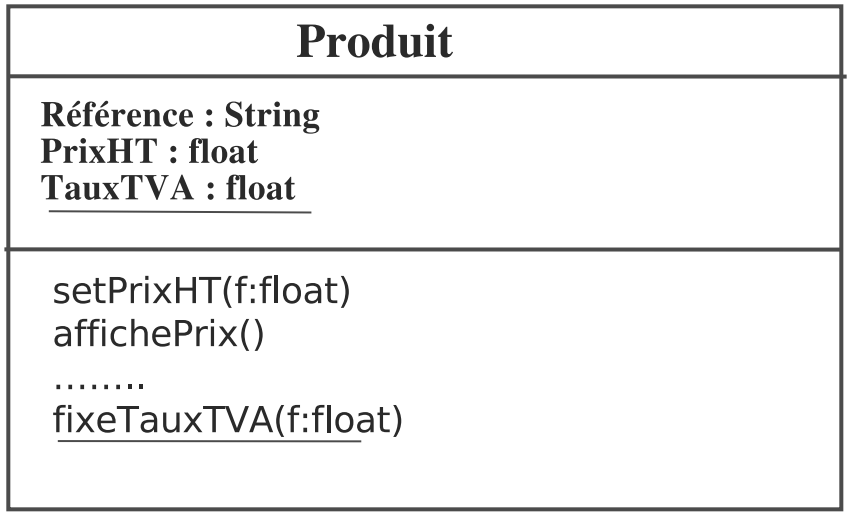


Figure : Opérations de classe

Constructeurs et destructeurs

Des opérations particulières

- Gestion de la durée de vie des instances
- Constructeur : création des instances
- Destructeur : destruction des instances

Constructeurs et destructeurs : notation

Notation

- stéréotypes <<create>> ou <<destroy>>
- stéréotypes : chaînes entre chevrons attachées aux éléments UML pour préciser la sémantique

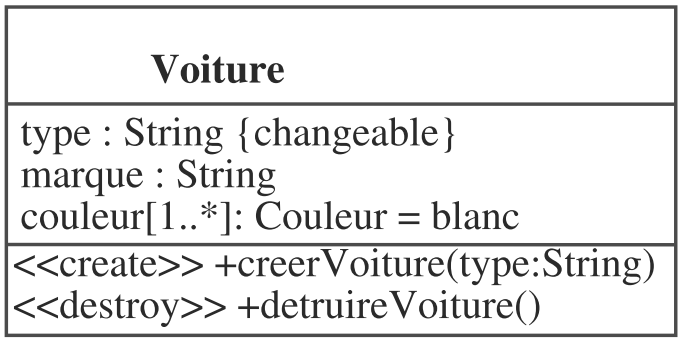


Figure : Constructeurs et destructeurs en UML

Le corps des opérations en UML

- langage d'action permettant de spécifier le comportement des opérations (Action Semantics dans Executable UML)
- Utilisation des diagrammes dynamiques pour spécifier le comportement des opérations
- Documentation avec du pseudo-code, dans une note de commentaire

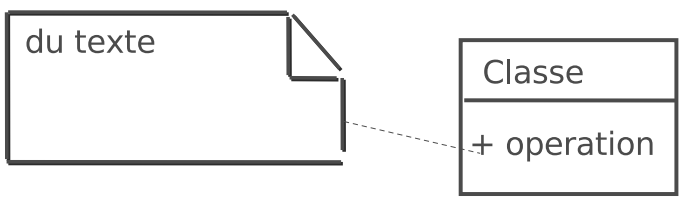


Figure : Note UML

Sommaire

Exemple

```
1  package ExemplesCours2;
2  public class Voiture
3  {
4      private String type;
5      private String marque;
6      private String couleur;
7      private static int nbrVoitures;
8      private static final int nbrRoues = 4;
9
10         public Voiture(String leType, String laMarque, String couleur){
11             type=leType;
12             marque=laMarque;
13             this.couleur=couleur;
14         }
15
16         public static int getNbrRoues(){
17             return nbrRoues;
18         }
19
20         public String getMarque(){
21             return marque;
22         }
23
24         private void setMarque(String m){
25             marque=m;
26         }
27
28         public void repeindre(String c){
29             couleur=c;
30         }
31 }
```

Déclaration de méthodes

- déclaration de constructeur : en Java, le constructeur doit avoir le même nom que la classe, et il n'y a pas de type de retour.
- pas vraiment de destructeur en Java. Il existe une méthode particulière nommée `finalize` qui est appelée quand le ramasse-miettes récupère l'espace alloué à l'objet car il n'est plus référencé.
- déclaration de méthode de classe : mot clef `static`. Appel : préfixer le nom de l'opération par le nom de la classe
- utilisation des mots clefs `private` et `public` pour définir la visibilité des méthodes
- il n'y a pas en Java la distinction entre paramètre in, out, ou inout.

Surcharge

Surcharge :

- dans une même classe
- plusieurs méthodes portant le même nom
- des signatures différentes

Exemple :

- une méthode `int add(int a, int b)`
- une méthode `float add(float a, float b)` .

Exemple

Listing 1 – Utilisation de la classe Voiture en java

```
1  package ExemplesCours2;
2  public class essaiVoiture{
3    public static void main(String[] arg){
4      Voiture v=new Voiture("C3", "Citroen", "rouge")
           ;
5      int nb=v.getNbrRoues();
6      System.out.println("Ma␣"+v.getMarque()+"␣a␣"+nb
         +"␣roues");
7    }
8  }
```

À noter ...

- la méthode étrange appelée `main` appelée par :

  ```
  > java ExemplesCours2.essaiVoiture
  ```

  - point d'entrée du programme
  - méthode statique : pas besoin de créer d'instance `essaiVoiture` pour utiliser la méthode `main`.
  - paramètre : arguments en ligne de commande (tableau de chaînes).
- la concaténation de chaînes pour l'affichage, et la traduction automatique d'entiers en chaînes.

Les accesseurs

Accès aux attributs, en lecture et en écriture
Par convention pour l'attribut `att` de type `T` :

```
1  T getAtt()
2  void setAtt(T valeur)
```

- `get` : statistiques sur les accès à l'attribut
- `set` : vérifications sur les valeurs, gestion des attributs dérivés

Les accesseurs – exemples

```
1  public class Point{
2      public static double dimension = 2 ;    //Variable de classe
3      private double x ;
4      private double y ;
5      public Point(){ //Constructeur par defaut
6          this(0,0) ;
7      }
8      public Point(double x , double y){ //Constructeur avec argument
9          this.setX(x) ;
10         this.setY(y) ;
11     }
12 //Accesseurs pour la variable x.
13     public double getX(){
14         return this.x ;
15     }
16     public void setX(double x){
17         this.x = x ;
18     }
19  //Accesseurs pour la variable y.
20     public double getY(){
21         return this.y ;
22     }
23     public void setY(double y){
24         this.y = y ;
25     }
26     public void symetrieSelonX(){
27         this.getY() = -this.getY() ;
28     }
29     public void symetrieSelonY(){
30         this.getX() = -this.getX() ;
31     }
32 }
```

Affectation, Déclaration de variables locales

- L'affectation se note =
- Dans le corps d'une méthode, on peut déclarer des variables locales. Par exemple :

```
int i;
int j=0;
Voiture v;
```

- Pas de visibilité : la portée de ces variables s'arrête à la fin de la méthode
- Pas de valeur initiale implicite

Création de nouvelles instances

Création : `new` + constructeur

```
Voiture v;
v=new Voiture("C4", "Citroen", "bleu");
```

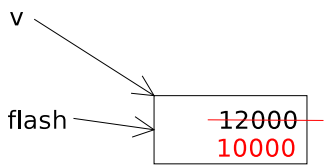
Le passage de paramètres

- Dans le corps d'une méthode, les paramètres sont comme des variables locales
- C'est comme si on avait des variables locales déclarées au début de la méthode, et qu'en début de méthode on affectait les valeurs des paramètres effectifs à ces variables locales
- Les paramètres de type simple (types de base en Java comme `int` et `boolean`, dont le nom commence par une minuscule) sont passés par valeur
- Tous les autres paramètres sont passés par référence (leur adresse est passée par valeur)

Illustration du passage de paramètres

```
public class Voiture{
  public int prix;
  public Voiture(int p){
   prix=p;
  }
}
```

```
public class Expertise{
  public void expertiser(v:Voiture){
    v.prix=10000;
 }
...
Voiture flash=new Voiture(12000);
...
expertiser(flash);
```



Pendant l'exécution de la méthode `expertiser`, `v` et `flash` désignent le même objet.

Figure : Passage de paramètres par référence

Exemple

Listing 2 – MonInt.java

```
1  package ExemplesCours2;
2  public class MonInt{
3
4    private int entier;
5
6    public MonInt(int e){
7      entier=e;
8    }
9
10   public int getEntier(){
11     return entier;
12   }
13
14   public void setEntier(int e){
15     entier=e;
16   }
17 }
```

Exemple

Listing 3 – Echange.java

```
1  package ExemplesCours2;
2  public class Echange{
3    public void fauxEchange(int a, int b){
4      System.out.println("a= "+a+" b= "+b);
5      int vi=a;
6      a=b;
7      b=vi;
8      System.out.println("a= "+a+" b= "+b);
9    }
10
11   public void pseudoEchange(MonInt a, MonInt b){
12     System.out.println("a.getEntier= "+a.getEntier()+" b.getEntier= "+b.
           getEntier());
13     int vi=a.getEntier();
14     a.setEntier(b.getEntier());
15     b.setEntier(vi);
16     System.out.println("a.getEntier= "+a.getEntier()+" b.getEntier= "+b.
           getEntier());
17   }
```

Exemple

Listing 4 – Echange.java

```
1     public static void main(String[] args){
2       int x=2, y=3;
3       System.out.println("x= "+x+" y= "+y);
4       Echange echange=new Echange();
5       echange.fauxEchange(x,y);
6       System.out.println("x= "+x+" y= "+y);
7
8       MonInt xx=new MonInt(2);
9       MonInt yy=new MonInt(3);
10      System.out.println("xx.getEntier= "+xx.getEntier()+" yy.getEntier= "
            +yy.getEntier());
11      echange.pseudoEchange(xx,yy);
12      System.out.println("xx.getEntier= "+xx.getEntier()+" yy.getEntier= "
            +yy.getEntier());
13    }
14
15  }
```

Test d'égalité sur des objets

- == teste l'égalité de référence
- la méthode equals, applicable sur n'importe quel objet, teste l'égalité de valeur

```
1  String s1="toto";
2  String s2="to";
3  s2+="to";
4  s1==s2; // false
5  s1.equals(s2); // true
```

Désignation de l'instance courante

- Receveur du message = instance courante
- Désigné par `this`
- Usage : conflit de noms ou utilisation directe

L'instruction `return`

- Retour d'un résultat
- Sortie immédiate de la méthode
- Cohérence entre valeur retournée et type de retour

Les commentaires

```
// ceci est un commentaire (s'arrête à la fin de la ligne)
/* ceci est un autre commentaire
   qui s'arrête quand on rencontre le marqueur de fin que voilà */
/** ceci est un commentaire particulier, utilisé par l'utilitaire javadoc **/
```

Affichage

On peut afficher des données sur la console grâce à une bibliothèque Java

```
System.out.println(‘‘affichage puis passage à la ligne’’);
System.out.print(‘‘affichage sans ’’);
System.out.print(‘‘passer à la ligne’’);
```

La méthode `toString()`

- Implicite dans toutes les classes
- Retourne une chaîne de caractères qui représente une instance ou son état sous une forme lisible et affichable
- Méthode par défaut : retourne une désignation de l'instance

La méthode `toString()` : exemple

Listing 5 – Personne.java

```
1  package ExemplesCours2;
2  public class Personne{
3    private String nom;
4    private int numSecu;
5
6    public String toString(){
7      String result=nom+" "+age;
8      return result;
9    }
10 }
```

Conditionnelle simple

Listing 6 – Conditionnelle en Java

```
1        if (expression booleenne) {
2          bloc1
3          }
4         else {
5          bloc2
6         }
```

- La condition doit être évaluable en true ou false et elle est obligatoirement entourée de parenthèses.
- Les points-virgules sont obligatoires après chaque instruction et interdits après }.
- Si un bloc ne comporte qu'une seule instruction, on peut omettre les accolades qui l'entourent.
- Les conditionnelles peuvent s'imbriquer.

Conditionnelle simple : exemple

Listing 7 – Conditionnelle en Java

```
1  int a =3;
2  int b =4;
3  System.out.print("Le plus petit entre "+a+" et "+b
       +" est : ");
4  if (b <a ) {
5    System.out.println(b);
6  }
7  else { System.out.println(a);
8  }
```

L'opérateur conditionnel ( ) ? ... : ...

Le : se lit *sinon*.

```
1  System.out.println( (b < a) ? b : a );
2  int c = (b < a) ? a-b : b-a ;
```

L'instruction de choix multiples

```
1  switch (expr entiere ou caractere) {
2  case i:
3  case j:
4  [bloc d'intructions]
5  break;
6  case␣k:
7  ...
8  default:
9  ...
10 }
```

- L'instruction `default` est facultative ; elle est à placer à la fin. Elle permet de traiter toutes les autres valeurs de l'expression n'apparaissant pas dans les cas précédents.
- Le `break` est obligatoire pour ne pas traiter les autres cas.

L'instruction de choix multiples : exemple

```
1  int mois, nbJours;
2  switch (mois) {
3   case 1:
4   case 3:
5   case 5:
6   case 7:
7   case 8:
8   case 10:
9   case 12:
10   nbJours = 31;
11   break;
12  case 4:
13  case 6:
14  case 9:
15  case 11:
16   nbJours = 30;
17   break;
18  case 2:
19    if ( ((annee % 4 == 0) && !(annee % 100 == 0)) || (annee % 400 == 0)
          )
20      nbJours = 29;
21    else
22      nbJours = 28;
23    break;
24  default nbJours=0;
25       }
26      }
```

Switch et énumérations

```
1  public enum Day {
2      SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
3      THURSDAY, FRIDAY, SATURDAY
4  }
5
6  public class EnumTest {
7      Day day;
8
9      public EnumTest(Day day) {
10         this.day = day;
11     }
12
13     public void tellItLikeItIs() {
14         switch (day) {
15             case MONDAY:
16                 System.out.println("Mondays are bad.");
17                 break;
18
19             case FRIDAY:
20                 System.out.println("Fridays are better.");
21                 break;
22
23             case SATURDAY: case SUNDAY:
24                 System.out.println("Weekends are best.");
25                 break;
26
27             default:
28                 System.out.println("Midweek days are so-so.");
29                 break;
30         }
31     }
32 }
```

while

Syntaxe :

```
1 while (expression) {
2   bloc
3 }
```

Exemple :

```
1 int max = 100, i = 0, somme = 0;
2 while (i <= max) {
3   somme += i;       // somme = somme + i
4   i++ ;
5 }
```

do while

Syntaxe :

```
1        do
2          { bloc }
3        while (expression)
```

```
1  int max = 100, i = 0, somme = 0 ;
2  do {
3   somme += i ;
4   i++;
5  }
6  while ( i <= max );
```

for

Syntaxe :

```
1 for ( expression1; expression2 ; expression3){
2     bloc
3 }
```

- utilisée pour répéter N fois un même bloc d'instructions
- `expression1` : initialisation. Précise en général la valeur initiale de la variable de contrôle (ou compteur)
- `expression2` : la condition à satisfaire pour rester dans la boucle
- `expression3` : une action à réaliser à la fin de chaque boucle. En général, on actualise le compteur.

for : exemple

```
1  int somme = 0, max = 100;
2  for (int i =0 ; i <= max ; i++ ) {
3   somme += i ;
4  }
```