Opérations UML et leur implémentation par des méthodes Java

Université de Montpellier
Faculté Des Sciences
Modélisation et Programmation par Objets (niveau 1)
PEIP

Classes, opérations et méthodes

Partie précédente

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

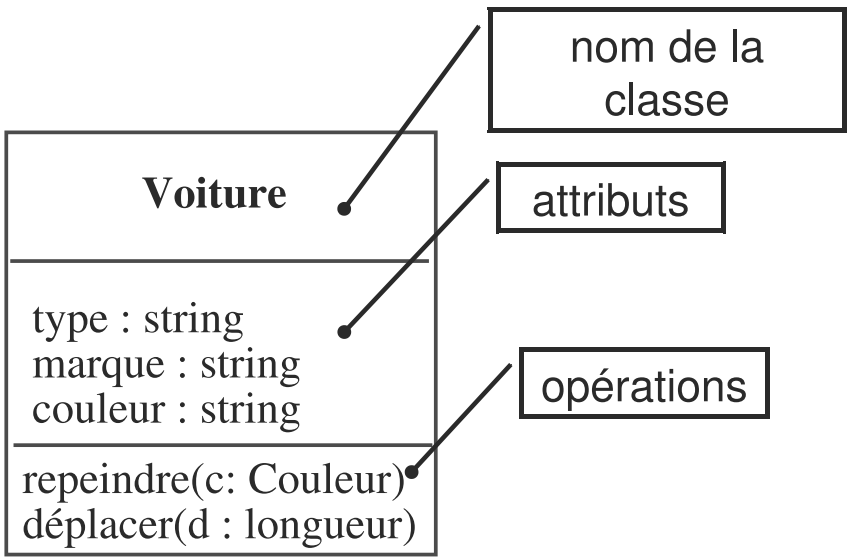
Méthodes et opérations

- 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

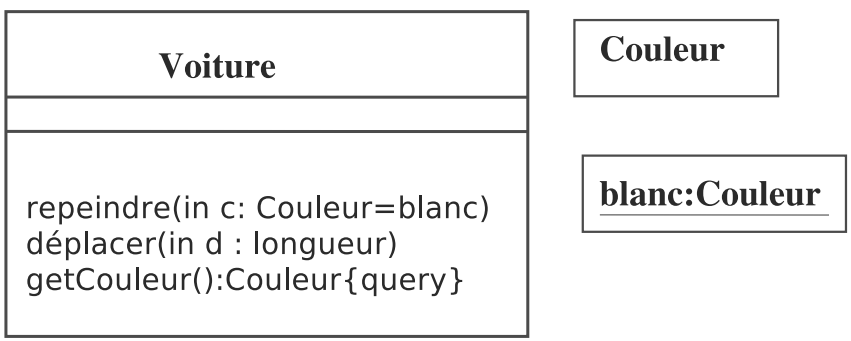
Opérations

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



Les éléments d'une opération

- nom
- visibilité
- paramètres avec leur direction
- type de retour et valeur par défaut
- des propriétés additionnelles



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 (voir avec le cours sur l'héritage)

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. Si rien n'est écrit, c'est *in*.

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.

- la `multiplicité` de chaque paramètre est définie par une valeur (1, 2, n, ...) ou une plage de valeurs (1..*, 1..6, ...).

Propriétés

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

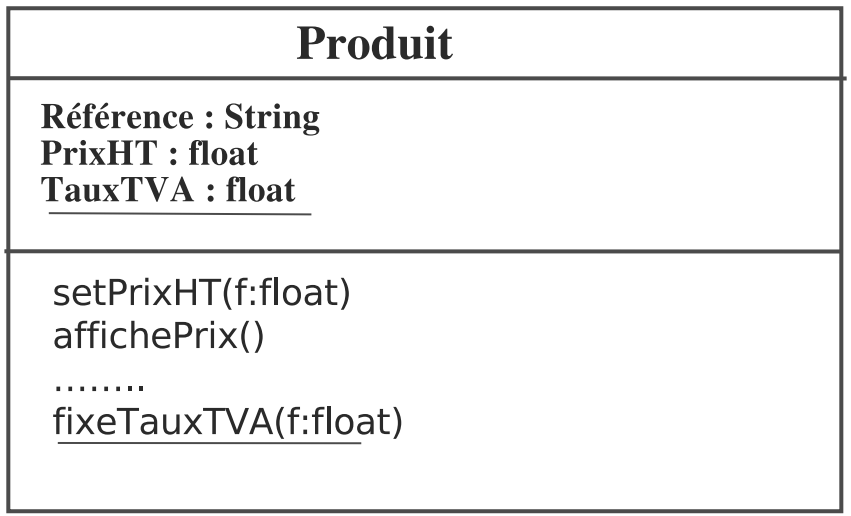
Exemples

Dans une classe TicketDeCaisse pour une chaîne de magasins bio

- +prixTotal() :float
- +nbArticles() :float {query}
- +exportPDF(in f :FormatTicket) :File
- +ajoutArticle(in a :Article, in qte :int)
- -articlesDeTVA(in taux :TVA) :Article[*]
- +arrondiSolidaire()

Opérations dont la portée est la classe

- C'est une opération qui ne s'applique pas à une instance.
- Elle peut être appelée sans avoir créé d'objet.



Constructeurs et destructeurs

Des opérations particulières

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

Notation

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

Le corps des opérations en UML

- Utilisation des diagrammes dynamiques pour spécifier le comportement des opérations
- Documentation avec du pseudo-code, dans une note de commentaire

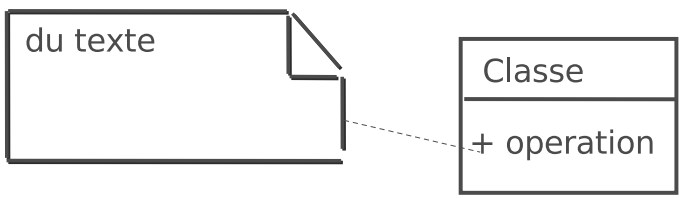
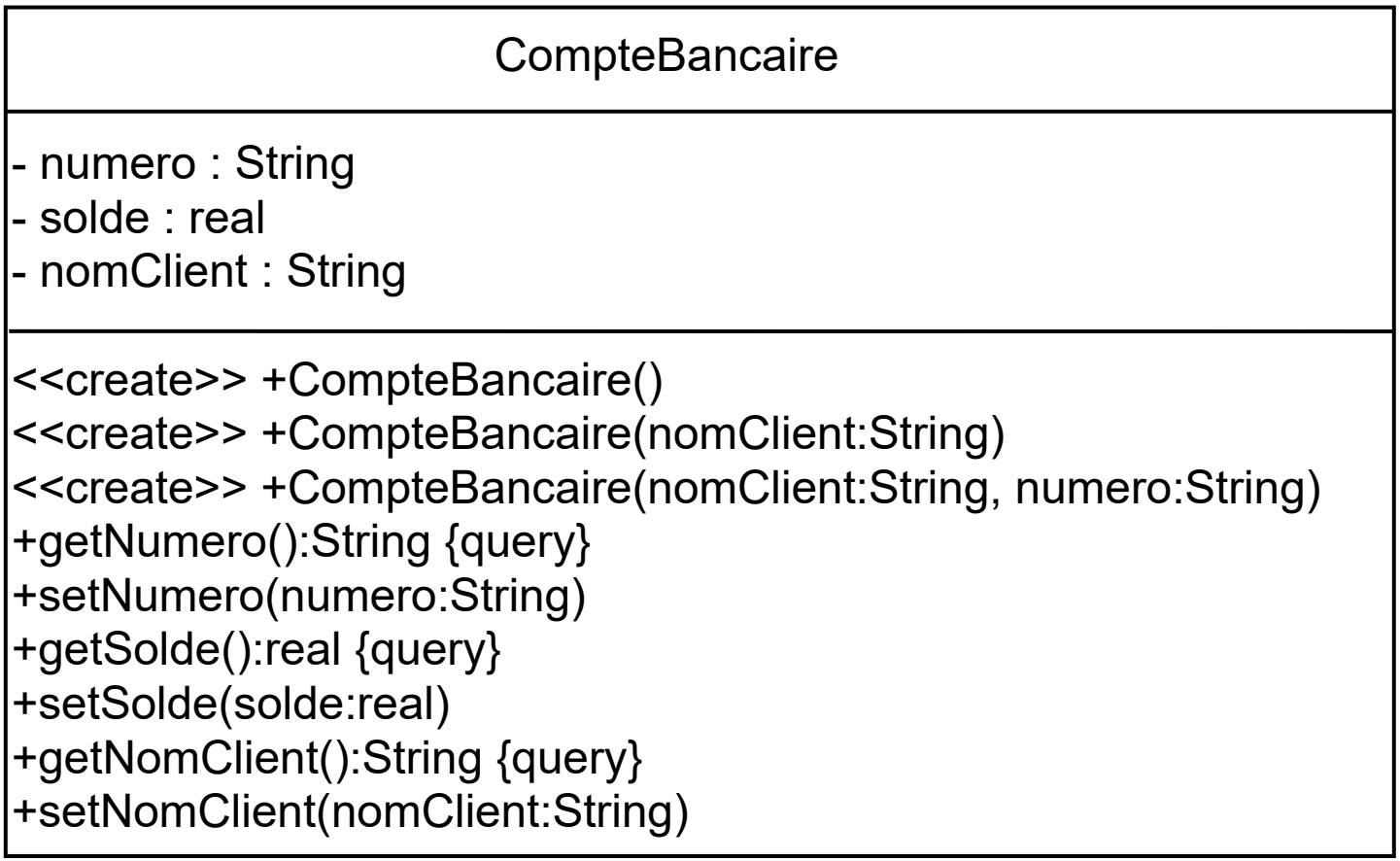


FIGURE – Note UML

Sommaire

Classe CompteBancaire



Classe CompteBancaire : rappel de la structure en Java

```
public class CompteBancaire {
        private String numero;
        private double solde;
        private String nomClient;
...}
```

Les méthodes en Java

Les algorithmes ou opérations propres aux classes sont appelés des méthodes.

Constitution :

- signature (entête)
  - visibilité
  - type de la valeur retournée (ou `void`)
  - nom
  - liste de paramètres (passés par valeur)
- corps
  - bloc d'instructions

```
public boolean debiter (double montant) { .... }
```

Les méthodes en Java

Une méthode (qui n'est pas static) s'applique à un objet ; on dit aussi que l'on envoie un message à l'objet.

Pendant l'exécution de la méthode, l'objet auquel le message est envoyé (auquel la méthode est appliquée) est désigné par la pseudo-variable this.

Constructeurs

Les constructeurs sont des méthodes qui servent à initialiser les objets au moment de leur création.

Particularités :

- ils n'ont pas de type de retour
- ils portent le même nom que la classe
- les valeurs des paramètres sont utilisées pour initialiser les attributs

Constructeur sans paramètre (par défaut)

Il initialise un objet (une instance) avec des valeurs par défaut, cohérentes avec la signification de la classe.

```
public class CompteBancaire{
...

   public CompteBancaire() {
       this.nomClient = "client inconnu";
       this.numero = "numéro non affecté";
       // on ne met rien dans solde,
       // la valeur par défaut nous convient
   }
...
}
```

Constructeur avec paramètre (constructeur 1)

Il initialise un objet (une instance) avec des valeurs passées en paramètre et si besoin complète avec des valeurs par défaut.

En voici un premier exemple : on initialise le nom du client avec une valeur passée en paramètre et le numéro avec une valeur par défaut.

```
public CompteBancaire(String nomClient) {
  this.nomClient = nomClient;
  this.numero = "numéro non affecté";
  // on ne met rien dans solde, la valeur par défaut nous convient
}
```

Constructeur avec paramètre (constructeur 2)

Il initialise un objet (une instance) avec des valeurs passées en paramètre et si besoin complète avec des valeurs par défaut.

En voici un second exemple : on initialise le nom du client et le numéro avec des valeurs passées en paramètres.

```
public CompteBancaire(String nomClient, String numero) {
  this.nomClient = nomClient;
  this.numero = numero;
    // on ne met rien dans solde, la valeur par défaut nous convient
}
```

Accesseurs

Ce sont des méthodes d'accès qui seront nécessaires pour manipuler les attributs (privés) depuis des méthodes hors de la classe.
Ils vont souvent par paires :

- un accesseur en lecture pour connaître la valeur ; sa forme est get suivi du nom de l'attribut commençant par une majuscule
- un accesseur en écriture pour modifier la valeur. Il permettra notamment de contrôler la manière dont on la modifie ; sa forme est set suivi du nom de l'attribut commençant par une majuscule

```
public String getNumero(){return this.numero; }

public void setNumero(String numero) {this.numero = numero; }

public String getNomClient(){return this.nomClient; }

public void setNomClient(String nomClient)
                {this.nomClient = nomClient; }
```

Accesseurs

L'accesseur en écriture sera utilisé pour contrôler les valeurs affectées aux attributs, en cohérence avec les règles métier.

- Le solde ne doit jamais être négatif (supposons que l'on interdise les comptes débiteurs pour cet exemple)

```
public void setSolde(double nouveauSolde) {
  if (nouveauSolde <0)
   {System.out.println("erreur : un solde ne doit pas être négatif");}
  else
   {this.solde = nouveauSolde;}
}
```

Accesseurs

```
public void setSolde(double nouveauSolde) {
  if (nouveauSolde <0)
   {System.out.println("erreur : un solde ne doit pas être négatif");}
  else
   {this.solde = nouveauSolde;}
}
```

Instruction conditionnelle à 2 branches

- si le solde passé en paramètre est négatif, on affiche une erreur
- sinon la valeur du paramètre est bien écrite dans l'attribut solde de l'objet receveur du message, désigné par this

Accesseurs

Une autre version de la même méthode (il y en aura une seule dans le programme, on devra choisir entre les deux).

```
public void setSolde(double nouveauSolde) {
  if (nouveauSolde  >= 0)
   {this.solde = nouveauSolde;}
}
```

Instruction conditionnelle à 1 branche

- si le solde passé en paramètre est positif la valeur du paramètre est bien écrite dans l'attribut solde de l'objet receveur du message, désigné par this

sous-entendu : on ne fait rien dans les autres cas

Autres méthodes

La méthode toString() est une méthode ordinairement présente dans toutes les classes.

Elle retourne une représentation de l'objet sous forme d'une chaîne de caractères (String).

```
public String toString() {
   return "Client : "+this.nomClient+
     " Numéro : "+this.numero+" solde = "+this.solde;
}
```

Autres méthodes

La méthode verseIntérêts a pour paramètre un taux d'intérêt (taux entre 0 et 1)

Elle modifie le solde en ajoutant des intérêts correspondant au taux

Elle ne retourne rien (void)

```
public void verseIntérêts(double taux) {
  this.solde = this.solde * (1+taux);
}
```

Appel de méthodes

```
public static void main(String[] args) {
   // Appel du constructeur 1
   CompteBancaire cb1 = new CompteBancaire("Marie");

   // Appel du constructeur 2
   CompteBancaire cb2 = new CompteBancaire("Sarah","ZZZ34");

   // Appel de la méthode toString
   System.out.println("cb1 "+cb1.toString());

   // on peut aussi écrire comme suit (appel implicite de toString)
   System.out.println("cb1 "+cb1);

....

// Appel de la méthode verseIntérêts
    cb1.verseIntérêts(0.01);
    cb2.verseIntérêts(0.05);
}
```

Appel de méthodes

```
public static void main(String[] args) {

    CompteBancaire cb1 = new CompteBancaire("Marie");
    CompteBancaire cb2 = new CompteBancaire("Sarah","ZZZ34");
    ....
    // Modification des soldes
    cb1.setSolde(400);
    cb2.setSolde(500);
    ....
}
```

A comprendre :

- pendant l'exécution de cb1.setSolde(400) ;
  this est cb1 ; l'attribut solde de cb1 est modifié
- pendant l'exécution de cb2.setSolde(500) ;
  this est cb2 ;l'attribut solde de cb2 est modifié

Synthèse sur les conditionnelles

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

Listing 1 – Conditionnelle en Java

- 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.

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 à choix multiples

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

- L'instruction `default` est facultative ; elle se place à la fin. Elle permet de traiter toutes les valeurs de l'expression n'apparaissant pas dans les cas précédents.
- Le `break` permet de 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 }
```

Sommaire

Synthèse

- Comprendre l'intérêt des classes qui regroupent structure et comportement d'un ensemble d'objets similaires
- Définition de la structure, composée d'attributs
- Définition du comportement, composé de méthodes
- Création d'objets
- Comprendre comment on appelle une méthode, par envoi de message à un objet (ou encore on dit application de la méthode à l'objet)
- Instructions conditionnelles