Le pattern visiteur

DUT M3105 : Conception et programmation objet avancées

Serge Rosmorduc
serge.rosmorduc@lecnam.net
Conservatoire National des Arts et Métiers

2016-2017

Le visiteur

Motivation

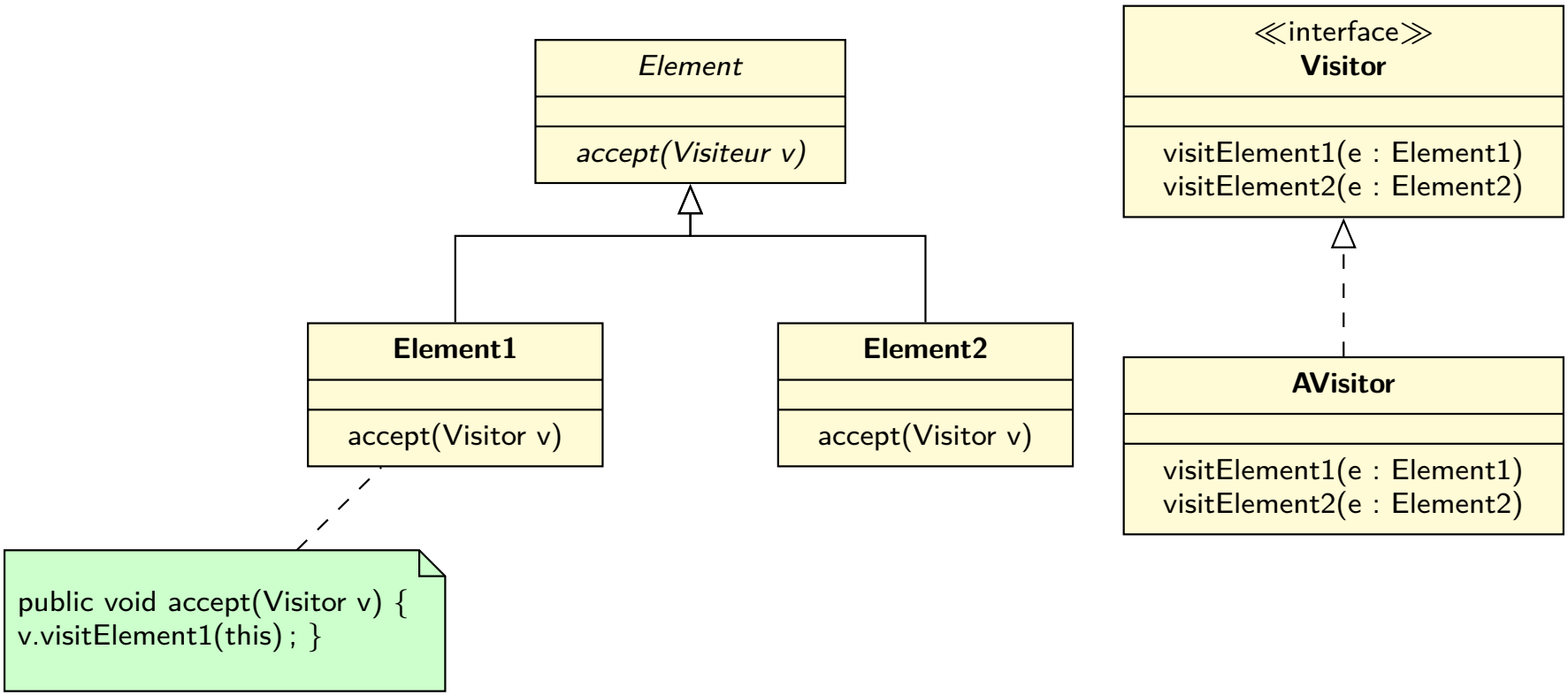
Permet de définir des actions sur une hiérarchie d'éléments, de manière externe à celle-ci, sans rompre l'encapsulation.

- Typiquement, on a un composite ;
- Pour ajouter de nouvelles manipulations :
  - on peut modifier le composite lui-même
  - pas pratique (et parfois impossible) si on a beaucoup d'actions à ajouter ;
  - ... ou utiliser un visiteur.

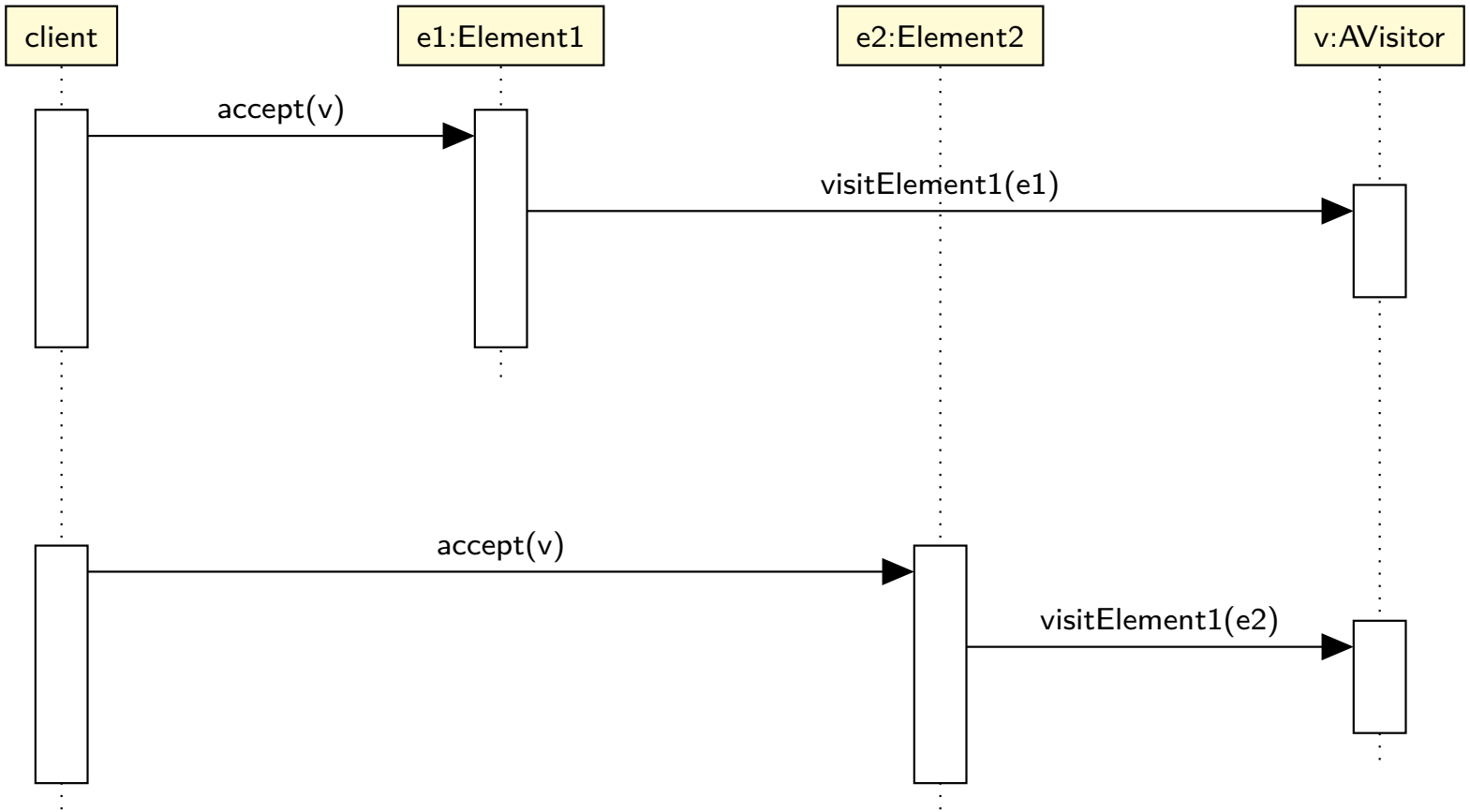
Le visiteur, motivation

- Un programme de dessin, avec un composite `Forme`, `Groupe`, `Ligne`, `Cercle` et `Texte` ;
- on définit (et on hérite) une méthode `dessiner` ;
- on veut ajouter des possibilités de sauvegarde du dessin, dans *plusieurs formats...* ;
- donc une méthode `writeAsSVG()` et une méthode `writeAsAdobeIllustrator()` ;
- si on les place dans la classe `Forme` et dans ses filles : gestion problématique si plusieurs programmeurs veulent ajouter des formats...
- on aimerait que la classe `Forme` se concentrât sur ses tâches propres.

Le visiteur : description



Visiteur (diagramme de séquence)



Le visiteur, implémentation java

```java
public abstract class Forme {
  // éventuellement des méthodes/variables d'instance
  // comme la couleur...
  public abstract void accept(FormeVisitor v);
}

public class Cercle extends Forme {
  private double x,y, r;  //...
  public void accept(FormeVisitor v) {
     v.visitCercle(this);
  }
}

public class Groupe extends Forme {
  private List<Forme> contenu; //...
  public void accept(FormeVisitor v) {
     v.visitGroupe(this);
  }
}
```

Le visiteur, implémentation java

```
1 public interface FormeVisitor {
2   void visitCercle(Cercle c);
3   void visitGroupe(Groupe g);
4   void visitLigne(Ligne l);
5   void visitTexte(Texte t);
6 }
```

Le visiteur, implémentation java

```java
public class Afficheur implements FormeVisitor {
  private void afficher(String s) {...}
  public void visitCercle(Cercle c) {
    afficher(c.getX()+","+ c.getY()+","+ c.getR());
  }
  public void visitGroupe(Groupe g) {
    for (Forme f: g.getContenu())
      f.accept(this);
  }
  public void visitLigne(Ligne l) {
    afficher(l.getPoint1() + "," l.getPoint2());
  }
  public void visitTexte(Texte t) {
    afficher(t.getTexte());
  }
}
```

Le visiteur, conséquences

- Ajouter les opérations directement dans le composite :
  - ajout d'un nouveau type d'élément : facile ;
  - ajout d'une nouvelle opération : complexe ;
- Utiliser un visiteur :
  - ajout d'un nouveau type d'élément : complexe ;
  - ajout d'une nouvelle opération : facile ;
- un visiteur est l'équivalent objet d'un switch case ;
- mais en mieux : on peut utiliser par exemple l'héritage (voir la suite) ;
- substitut pour les « types sommes » des langages fonctionnels, OCaml, Scala (case classes) ;
- pas mal de variantes possibles.

Visiteur et valeur de retour

Comment faire pour renvoyer une valeur avec un visiteur ?
Solution traditionnelle : faire de la valeur de retour une variable d'instance du visiteur ;

```
class SurfaceVisitor implements FormeVisitor {
  double surface= 0;
  public double getSurface() {return surface;}
  public void visitCercle(Cercle c) {
    surface+= Math.PI*r*r;
  }
  public void visitGroupe(Groupe g) {
    for (Forme f: g.getContenu()) {
      f.accept(this);
    }
  }
  public void visitLigne(Ligne l) {/*RIEN*/}
  public void visitTexte(Texte t) {/*RIEN*/}
}
```

Visiteur et valeur de retour (utilisation)

```
1 Forme maForme= ......;
2 SurfaceVisitor v= new SurfaceVisitor();
3 // v à usage unique ?
4 maForme.accept(v);
5 System.out.println("Surface " + v.getSurface());
```

Version « moderne » avec les génériques

```
// V : type retourné.
// Utiliser Void pour les visiteurs "normaux"
public interface FormeVisitor<V> {
  V visitCercle(Cercle c);
  V visitGroupe(Groupe g);
  V visitLigne(Ligne l);
  V visitTexte(Texte t);
}
```

Version générique

```
class SurfaceVisitor implements FormeVisitor<Double> {
  public Double visitCercle(Cercle c) {
    return Math.PI*r*r;
  }
  public Double visitGroupe(Groupe g) {
    double r=0.0;
    for (Forme f: g.getContenu()) {
      r+= f.accept(this);
    }
    return r;
  }
  public Double visitLigne(Ligne l) {return 0;}
  public Double visitTexte(Texte t) {return 0;}
}
```

Visiteur et valeur de retour (utilisation)

```
1 Forme maForme= ......;
2 double s= maForme.accept(new SurfaceVisitor());
3 System.out.println("Surface " + s);
```

Version générique sans valeur de retour ???

```
public class Afficheur implements FormeVisitor<Void> {
  private void afficher(String s) {...}
  public Void visitCercle(Cercle c) {
    afficher(c.getX()+","+ c.getY()+","+ c.getR());
    return null;
  }
  public Void visitGroupe(Groupe g) {
    for (Forme f: g.getContenu())
      f.accept(this);
    return null;
  }
  public Void visitLigne(Ligne l) {
    afficher(l.getPoint1() + "," l.getPoint2());
    return null;
  }
  public Void visitTexte(Texte t) {
    afficher(t.getTexte());
    return null;
  }
}
```

Visiteurs et classes abstraites

note : on utilise ici la forme classique des visiteurs, sans génériques
Pour les cas où on ne veut pas devoir définir *toutes* les méthodes du visiteur :

```
public class AbstractFormeVisitor implements FormeVisitor {
  public void visitCercle(Cercle c) {/*rien*/}
  public void visitGroupe(Groupe g) {/*rien*/}
  public void visitLigne(Ligne l) {/*rien*/}
  public void visitTexte(Texte t){/*rien*/}
}
```

- la classe peut être abstraite si on veut ;
- un visiteur concret peut l'étendre en redéfinissant uniquement les méthodes nécessaires ;
- si on ajoute une classe à la hiérarchie (et une méthode au visiteur) peut limiter l'ampleur des modifications nécessaires.

Visiteurs et composites

Un visiteur abstrait peut comporter un parcours par défaut des composites :

```
public abstract class DeepFormeVisitor
                    implements FormeVisitor {
  public void visitGroupe(Groupe g) {
    for (Forme f: g.getContenu())
      f.accept(this);
  }
}
```

Visiteurs et composites

Les visiteurs concrets pourront l'étendre en disposant « gratuitement » du parcours :

```
class SurfaceVisitor extends DeepFormeVisitor {
  double surface= 0;
  public double getSurface() {return surface;}
  public void visitCercle(Cercle c) {
    surface+= Math.PI*r*r;
  }
  public void visitLigne(Ligne l) {/*RIEN*/}
  public void visitTexte(Texte t) {/*RIEN*/}
  // pas de visitGroupe : hérité !
}
```

Visiteurs, un exemple dans un logiciel existant...