Visiteurs

Alexis Nasr
Franck Dary
Pacôme Perrotin

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

Idée générale

- L'objectif des visiteurs est d'ajouter une nouvelle fonctionnalité à une classe *C* (presque) sans modifier son contenu.
- On ajoute à *C* la méthode `accept(Visitor visitor)` qui accepte la classe `Visitor`.
- La méthode `accept` appelle la méthode `visit` de la classe `Visitor`
- La méthode `visit` constitue la fonctionnalité ajoutée.
- Le code correspondant se trouve dans la classe `Visiteur` et non dans la classe *C* !

Exemple

- La classe abstraite `Forme`

```
public abstract class Forme{}
```

- La classe `Cercle`

```
public class Cercle extends Forme{
    private double rayon;
    public Cercle(double rayon){this.rayon = rayon;}
    double getRayon(){return rayon;}
}
```

- La classe `Carre`

```
public class Carre extends Forme{
    private double cote;
    public Carre(double cote){this.cote = cote;}
    double getCote(){return cote;}
}
```

On rend les classes visitables

- La classe abstraite `Forme`

```
public abstract class Forme{
      public abstract double accept(Visitor visitor);
}
```

- La classe `Cercle`

```
public class Cercle extends Forme{
    private double rayon;
    public Cercle(double rayon){this.rayon = rayon;}
    double getRayon(){return rayon;}
    public double accept(Visitor visitor){
        return visitor.visit(this);
    }
}
```

- La classe `Carre`

```
public class Carre extends Forme{
    private double cote;
    public Carre(double cote){this.cote = cote;}
    double getCote(){return cote;}
    public double accept(Visitor visitor){
        return visitor.visit(this);
    }
}
```

L'interface `Visitor` et une implémentation

- L'interface `Visitor`

```
public interface Visitor {
    public double visit(Carre carre);
    public double visit(Cercle cercle);
}
```

- La classe `Aire` implémente l'interface

```
public class Aire implements Visitor {

    public double visit(Carre carre){
        return carre.getCote() * carre.getCote();
    }
    public double visit(Cercle cercle){
        return Math.PI * cercle.getRayon() * cercle.getRayon();
    }
}
```

Visiteur générique

On aimerait que la méthode `visit` puisse retourner autre chose qu'un `double`

- L'interface `Visitor`

```
public interface Visitor <T> {
    public T visit(Carre carre);
    public T visit(Cercle cercle);
}
```

- La classe `Aire` implémente l'interface

```
public class Aire implements Visitor <Double> {

    public Double visit(Carre carre){
        return carre.getCote() * carre.getCote();
    }
    public Double visit(Cercle cercle){
        return Math.PI * cercle.getRayon() * cercle.getRayon();
    }
}
```

Modifications de la méthode accept

- La classe abstraite `Forme`

```
public abstract class Forme{
      public abstract <T> T accept(Visitor <T> visitor);
}
```

- La classe `Cercle`

```
public class Cercle extends Forme{
    private double rayon;
    public Cercle(double rayon){this.rayon = rayon;}
    double getRayon(){return rayon;}
    public <T> T accept(Visitor <T> visitor){
        return visitor.visit(this);
    }
}
```

- La classe `Carre`

```
public class Carre extends Forme{
    private double cote;
    public Carre(double cote){this.cote = cote;}
    double getCote(){return cote;}
    public <T> T accept(Visitor <T> visitor){
        return visitor.visit(this);
    }
}
```

On peut définir d'autres visiteurs

```
public class NomFr implements Visitor <String> {

    public String visit(Carre carre){
        return "CARRE";
    }
    public String visit(Cercle cercle){
        return "CERCLE";
    }

}
```

Pas besoin de modifier les classes `Forme`, `Carre` et `Cercle`.

Utilisation

```
public class Main{
    public static void main(String[] args) {
        ArrayList<Forme> listeFormes = new ArrayList<Forme>();
        listeFormes.add(new Carre(10));
        listeFormes.add(new Carre(20));
        listeFormes.add(new Carre(30));
        listeFormes.add(new Cercle(10));
        listeFormes.add(new Cercle(20));
        listeFormes.add(new Cercle(30));

        Aire aire = new Aire();
        NomFr nomFr = new NomFr();

        for(Forme forme : listeFormes){
            System.out.println("aire " + forme.accept(nomFr) + " = "+ forme.accept(aire));
        }

    }
}
```

Exécution

```
alexis@corail3:~/test_java/visiteur$ java Main
aire CARRE = 100.0
aire CARRE = 400.0
aire CARRE = 900.0
aire CERCLE = 314.0
aire CERCLE = 1256.0
aire CERCLE = 2826.0
```