

# DESIGN PAT**T**ERN COMMAND

RAFIKI Younes – BELDENT Corentin – LEGER Charlie –  
BOURIAUD Thomas

M3 I05 – Conception & programmation objet avancées

# SOMMAIRE

- Qu'est ce qu'un design pattern ?
- Pattern Command
  - Utilité du pattern command
  - Diagramme UML
  - Conséquences
  - Exemple d'utilisation

## DESIGN PATTERN - RAPPEL

- Solution fréquente à un problème récurrent
- Défini par un :
  - Nom
  - Problématique
  - Solution
  - Conséquences

# PROBLEMATIQUE

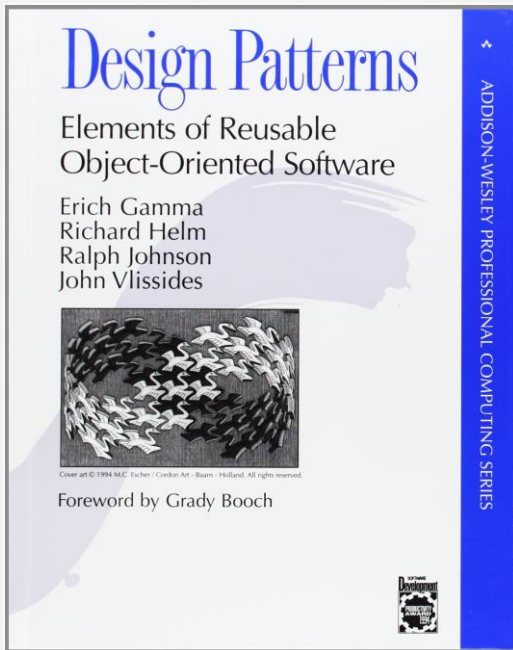
- Comment effectué des requêtes sur des objets sans ne rien connaitre des opérations nécessaires ?
  - Pour cela, nous avons le pattern **COMMAND**

# PATTERN COMMAND

- Un pattern comportemental
- Isoler une requête
- Souvent utilisé dans des interfaces graphiques (Tool bar ou bouton)
- Bibliothèque SWING



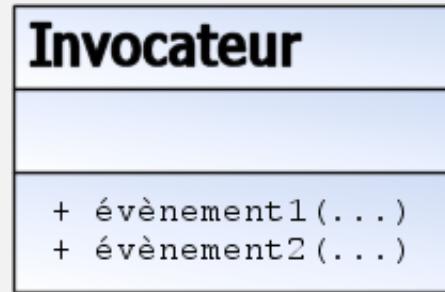
# UTILITÉ DU PATTERN COMMAND



- Encapsuler une requête sous la forme d'un objet
- Tracer la requête
- Annuler des opérations « Undo »
- « Logs » de requêtes
- File d'attente de requêtes

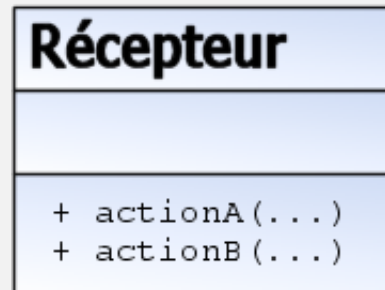
# UML

Déclenche  
l'exécution des  
commandes  
(Elément d'un  
menu IHM)



Abstraction d'un  
action, la requête est  
encapsulée

Reçoit la  
commande et  
réalise les  
opérations associés



appelle



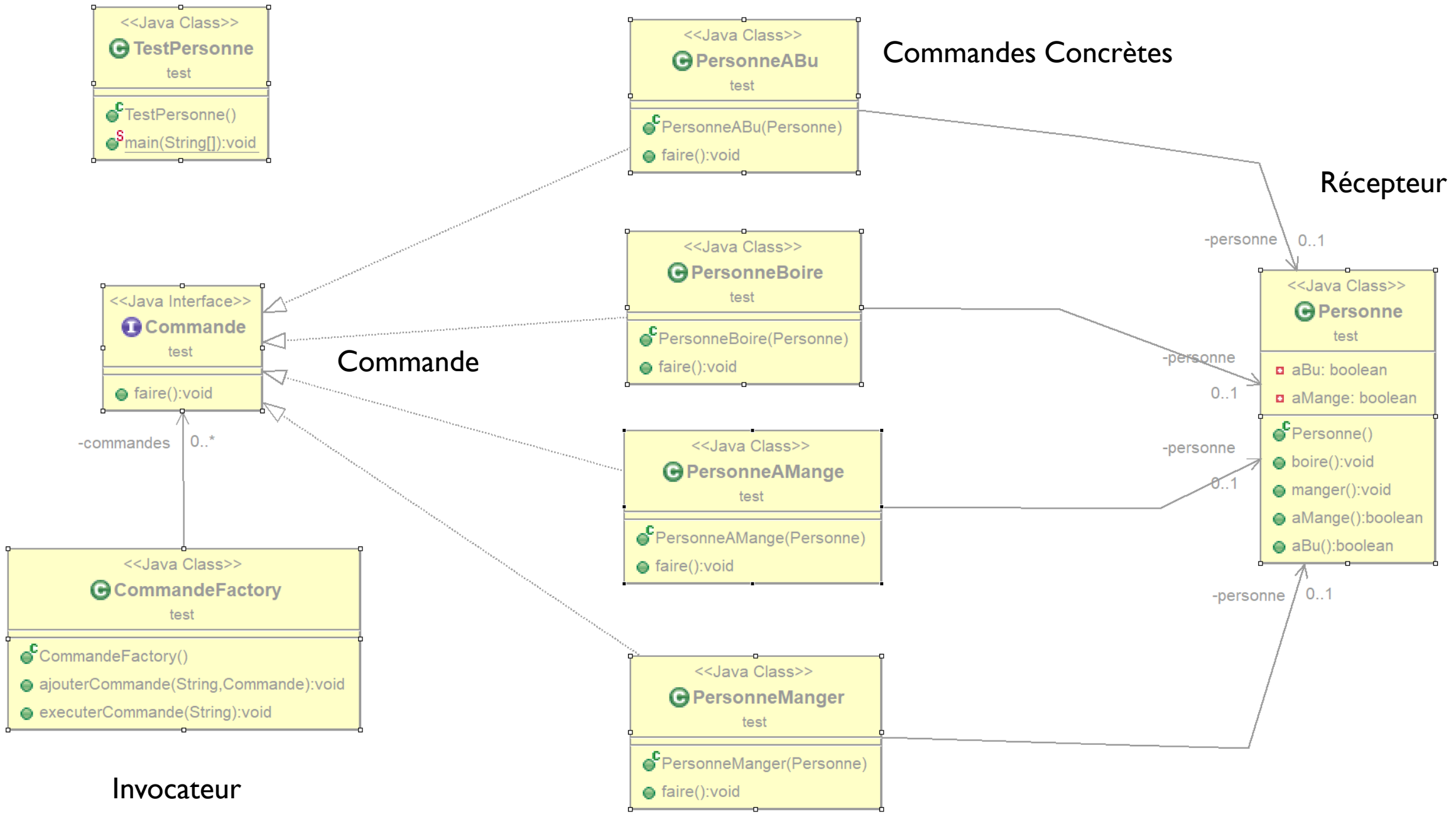
Implémente la  
méthode execute()  
Elle appelle des  
méthodes de  
l'objet Récepteur

# CONSÉQUENCES

- Défaire plusieurs niveaux : `undo()`
- Comportement transactionnel : `rollback()`
- Création et enregistrement de macros
- Etc ...



EXEMPLE



```

3 public class Personne {
4     private boolean aBu;
5     private boolean aMange;
6
7     public Personne() {
8         this.aBu = false;
9         this.aMange = false;
10    }
11
12    public void boire() {
13        this.aBu = true;
14    }
15
16    public void manger() {
17        this.aMange = true;
18    }
19
20    public boolean aMange() {
21        return this.aMange;
22    }
23
24    public boolean aBu() {
25        return this.aBu;
26    }
27 }

```

```

3 import java.util.HashMap;
4
5 public class CommandeFactory {
6     private final HashMap<String, Commande> commandes;
7
8     public CommandeFactory() {
9         this.commandes = new HashMap<>();
10    }
11
12    public void ajouterCommande(String nom, Commande commande) {
13        this.commandes.put(nom, commande);
14    }
15
16    public void executerCommande(String nom) {
17        if (this.commandes.containsKey(nom))
18            this.commandes.get(nom).faire();
19    }
20
21 }

```

```
3 @FunctionalInterface
4 public interface Commande {
5     public void faire();
6 }
```

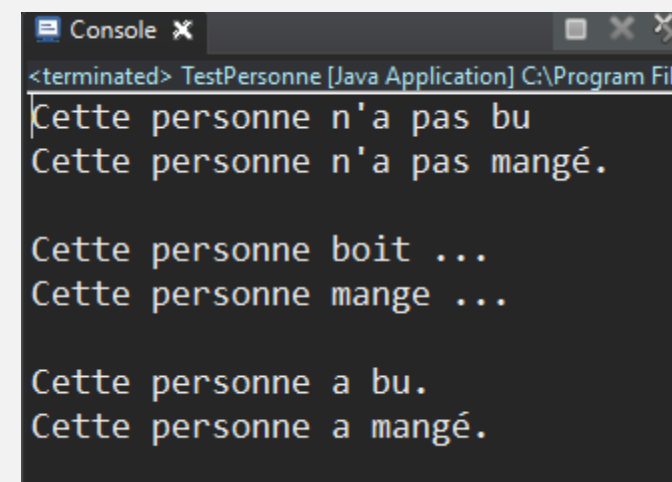
```
3 public class PersonneBoire implements Commande {
4
5     private Personne personne;
6
7     public PersonneBoire(Personne personne) {
8         this.personne = personne;
9     }
10
11     @Override
12     public void faire() {
13         this.personne.boire();
14         System.out.println("Cette personne boit ...");
15     }
16
17 }
```

```
3 public class PersonneABu implements Commande {
4
5     private Personne personne;
6
7     public PersonneABu(Personne personne) {
8         this.personne = personne;
9     }
10
11     @Override
12     public void faire() {
13         if (this.personne.aBu())
14             System.out.println("Cette personne a bu.");
15         else
16             System.out.println("Cette personne n'a pas bu");
17     }
18
19 }
```

```

3 public class TestPersonne {
4
5     public static void main(String[] args) {
6         // Création d'une personne
7         Personne personne = new Personne();
8
9         // Création d'une commandeFactory
10        CommandeFactory cf = new CommandeFactory();
11
12        // Ajout des commandes
13        cf.ajouterCommande("Manger", new PersonneManger(personne));
14        cf.ajouterCommande("Boire", new PersonneBoire(personne));
15        cf.ajouterCommande("A bu", new PersonneABu(personne));
16        cf.ajouterCommande("A mangé", new PersonneAMange(personne));
17
18        // Exécution des commandes
19        cf.executerCommande("A bu");
20        cf.executerCommande("A mangé");
21
22        System.out.println();
23
24        cf.executerCommande("Boire");
25        cf.executerCommande("Manger");
26
27        System.out.println();
28
29        cf.executerCommande("A bu");
30        cf.executerCommande("A mangé");
31    }
32
33 }

```



Console x

<terminated> TestPersonne [Java Application] C:\Program Fi

Cette personne n'a pas bu  
 Cette personne n'a pas mangé.

Cette personne boit ...  
 Cette personne mange ...

Cette personne a bu.  
 Cette personne a mangé.

## SOURCE

- <http://zenika.developpez.com/tutoriels/java/patterns-command/>
- [https://fr.wikibooks.org/wiki/Patrons\\_de\\_conception/Commande#Utilisations](https://fr.wikibooks.org/wiki/Patrons_de_conception/Commande#Utilisations)
- [http://rpouiller.developpez.com/tutoriel/java/design-patterns-gang-of-four/?page=page\\_4#LVI-B](http://rpouiller.developpez.com/tutoriel/java/design-patterns-gang-of-four/?page=page_4#LVI-B)