

Développement en JAVA

Architecture et Pattern

Objectif

+
o

•

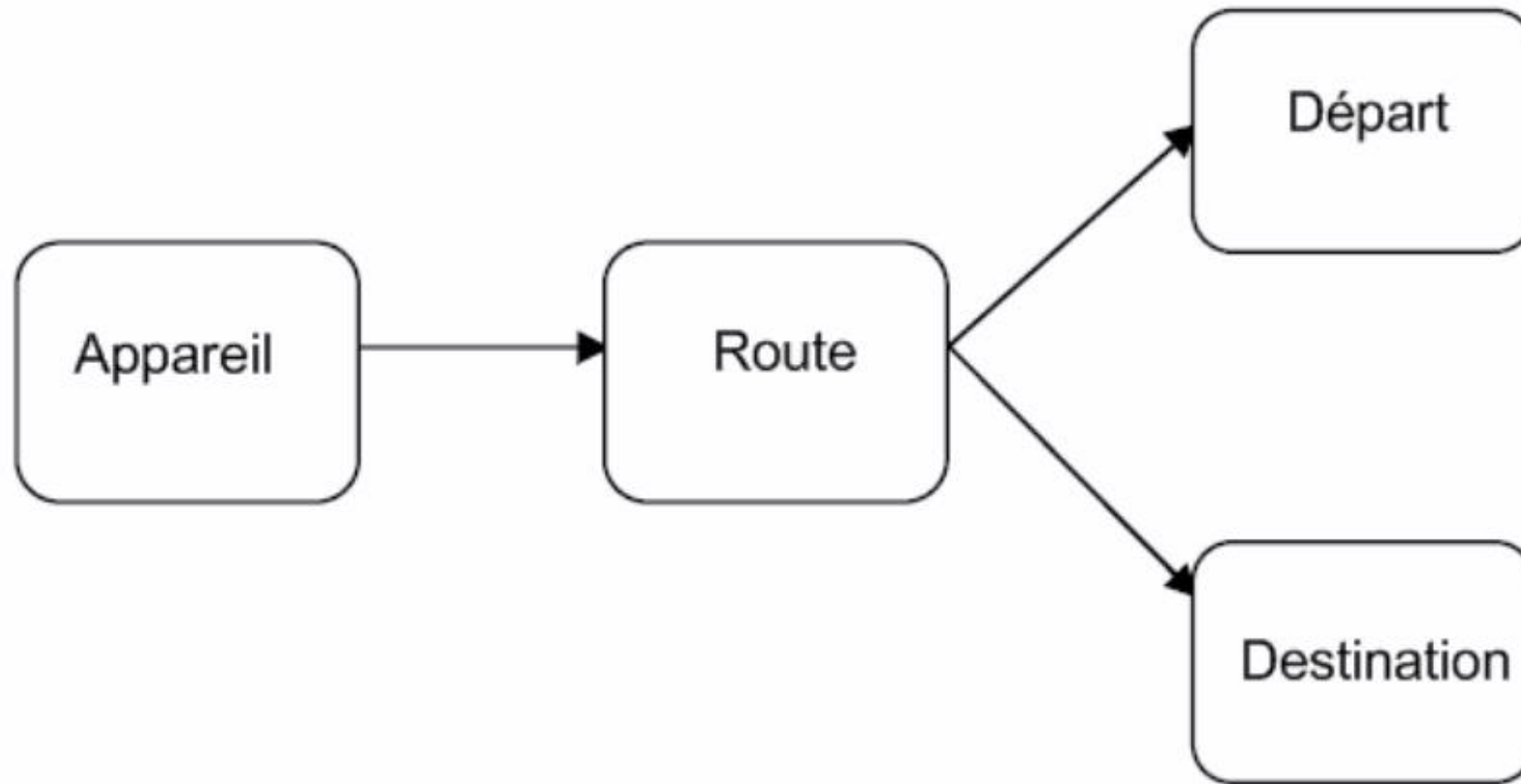
Modéliser une solution



Modéliser une solution



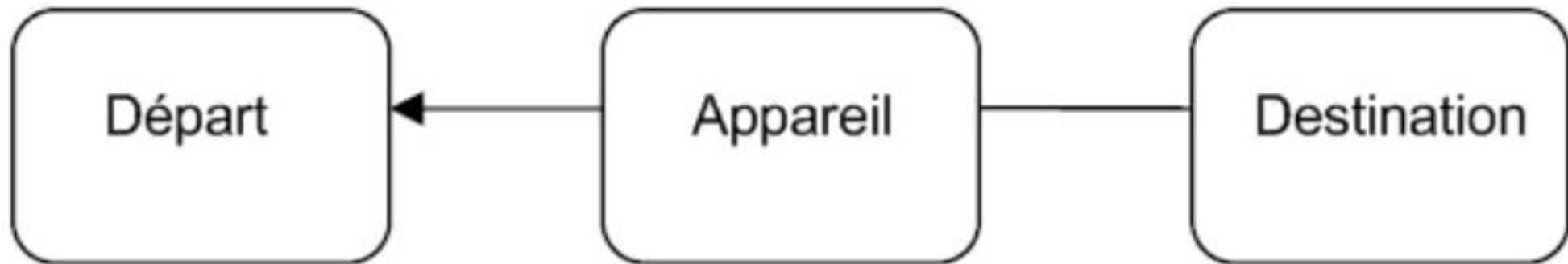
L'objet



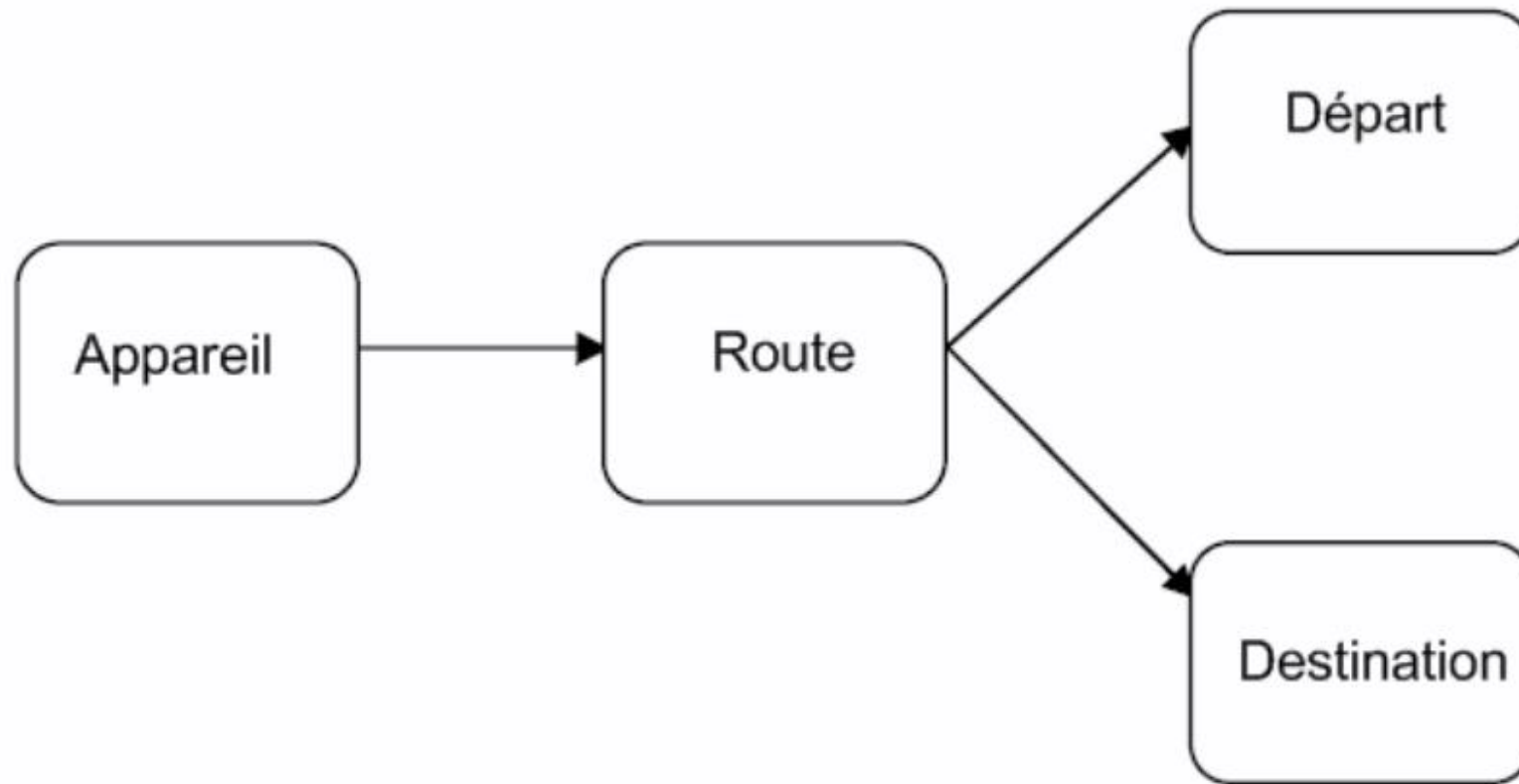
Modéliser un monde complexe



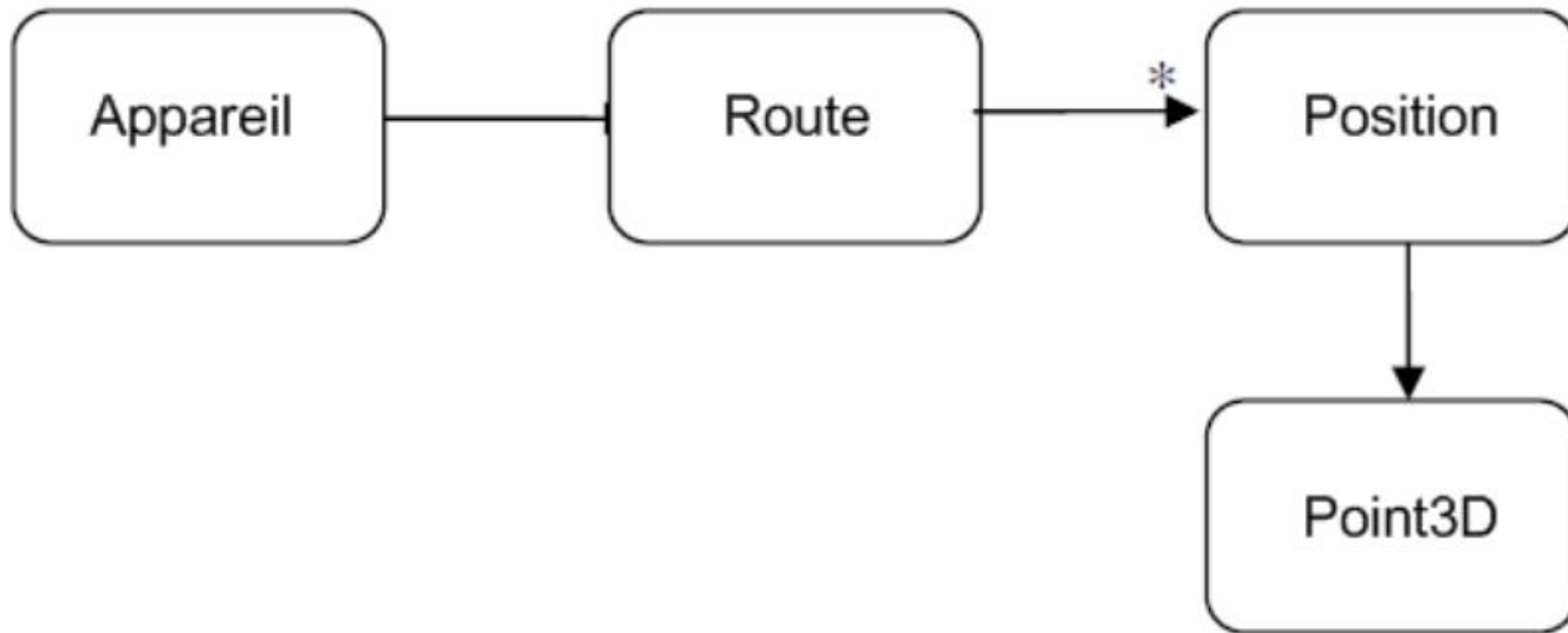
DDD vite fait (Domain Driven Design)



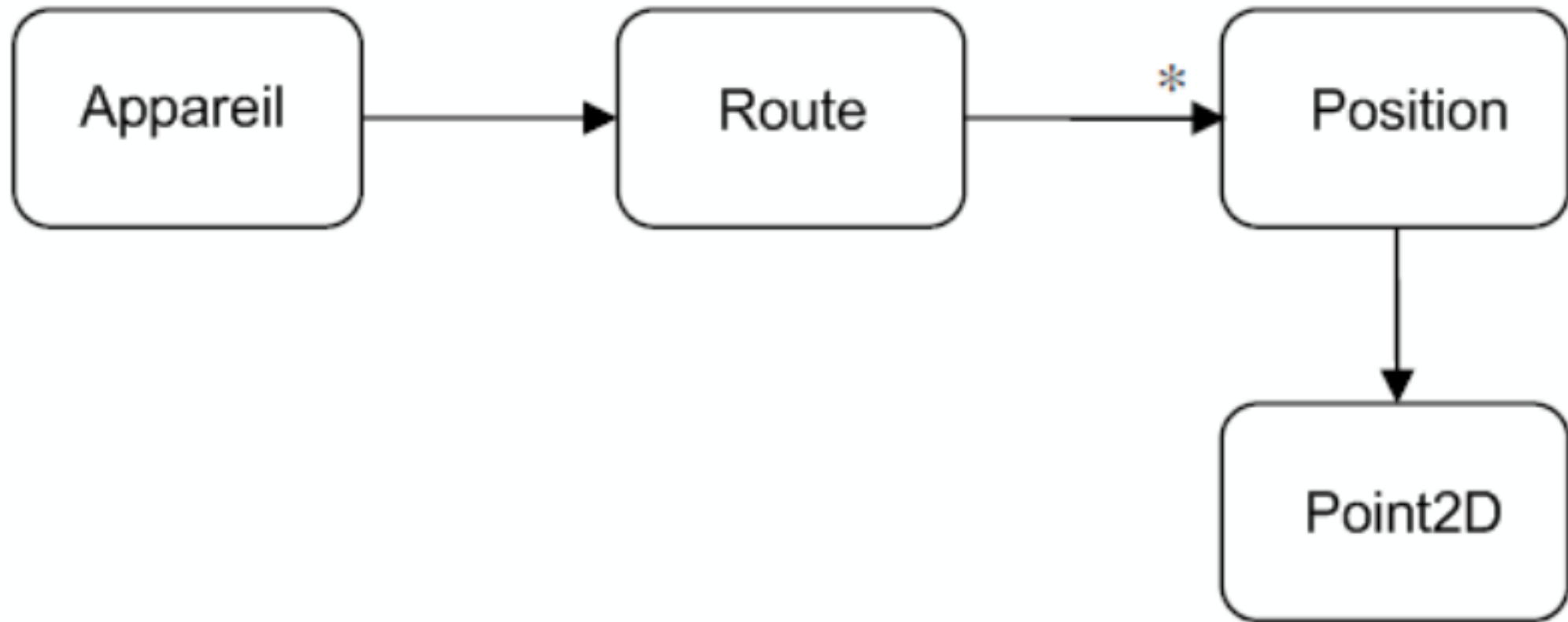
DDD vite fait (Domain Driven Design)



DDD vite fait (Domain Driven Design)

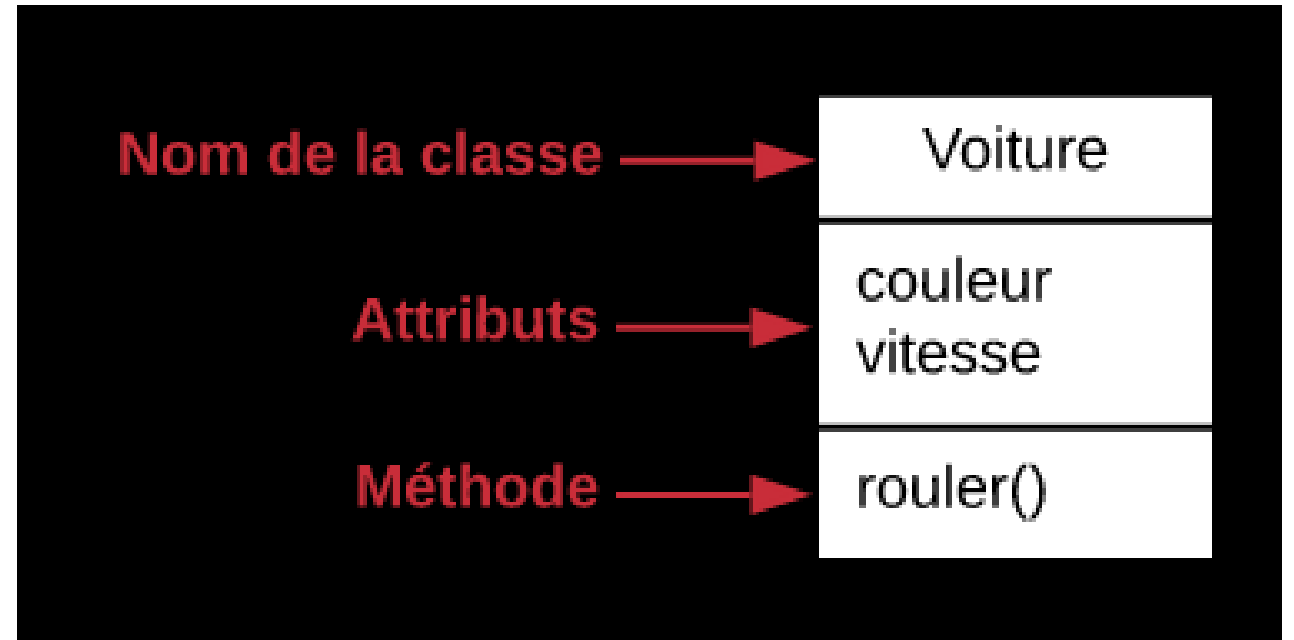


DDD vite fait (Domain Driven Design)



Responsabilité d'une entité

- Une instance d'objet gère son état.
- Personne d'autre que l'instance ne gère son état.



Contrôler ce modèle





Problème

Comment je gère le cas où deux voitures entrent en collision?

Comment je gère les transferts d'argent entre deux objets « comptes bancaires? »


Solution 1


```
Execute | > Share | Source File | STDIN
1 public class HelloWorld{
2
3     public static void main(String []args){
4         Voiture voitureUn = new Voiture();
5         Voiture voitureDeux = new Voiture();
6
7         voitureUn.colision(voitureDeux);
8     }
9
10
11 }
12
13 class Voiture {
14
15
16     public void colision(Voiture voitureDeux){
17         // traitement de la colision pour la voitureUn
18         voitureDeux.colision(this);
19     }
20 }
```


Solution 2


```
Execute | Share Source File STDIN
1 public class HelloWorld{
2
3     public static void main(String []args){
4         Voiture voitureUn = new Voiture();
5         Voiture voitureDeux = new Voiture();
6
7         voitureUn.colision(voitureDeux);
8     }
9
10
11 }
12
13 class Voiture {
14
15     public int etat = 100;
16
17     public void colision(Voiture voitureDeux){
18         this.etat = 50;
19         voitureDeux.etat = 50;
20     }
21 }
```


Solution 2

 Execute

 Share

 Source File

 STDIN

```
1 public class HelloWorld{
2
3     public static void main(String []args){
4         Voiture voitureUn = new Voiture();
5         Voiture voitureDeux = new Voiture();
6
7         voitureUn.colision(voitureDeux);
8     }
9
10
11 }
12
13 class Voiture {
14
15
16     public void colision(Voiture voitureDeux){
17         // traitement de la colision pour la voitureUn
18         voitureDeux.colision(this);
19     }
20 }
```

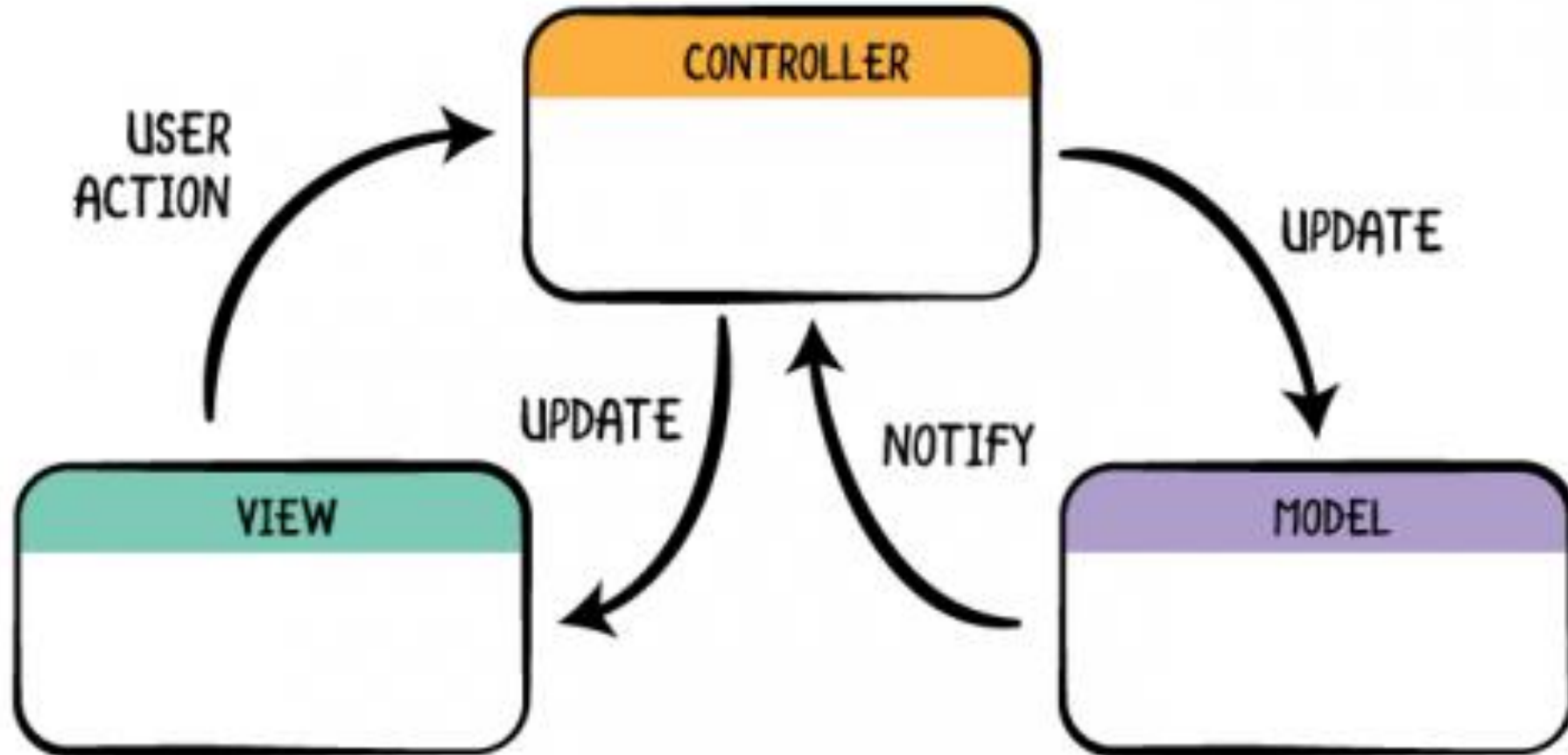
Result

[illegible]

Solution 3

```
Execute | Share | Source File | STDIN
1 public class HelloWorld{
2
3     public static void main(String []args){
4         Voiture voitureUn = new Voiture();
5         Voiture voitureDeux = new Voiture();
6
7         RouteManager.colision(voitureUn, voitureDeux);
8     }
9
10
11 }
12
13 class RouteManager{
14     static void colision(Voiture voitureUn, Voiture voitureDeux){
15         voitureUn.colision();
16         voitureDeux.colision();
17     }
18 }
19
20 class Voiture {
21
22     public int etat = 100;
23
24     public void colision(){
25         this.etat = 50;
26     }
27 }
```

Controller

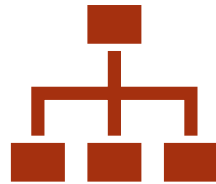


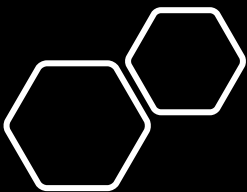
Controller



Pattern strategy

Une approche et des modèles de solutions adaptés à des problèmes
fréquemment rencontrés

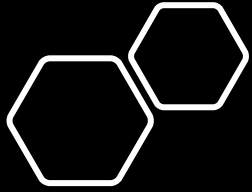




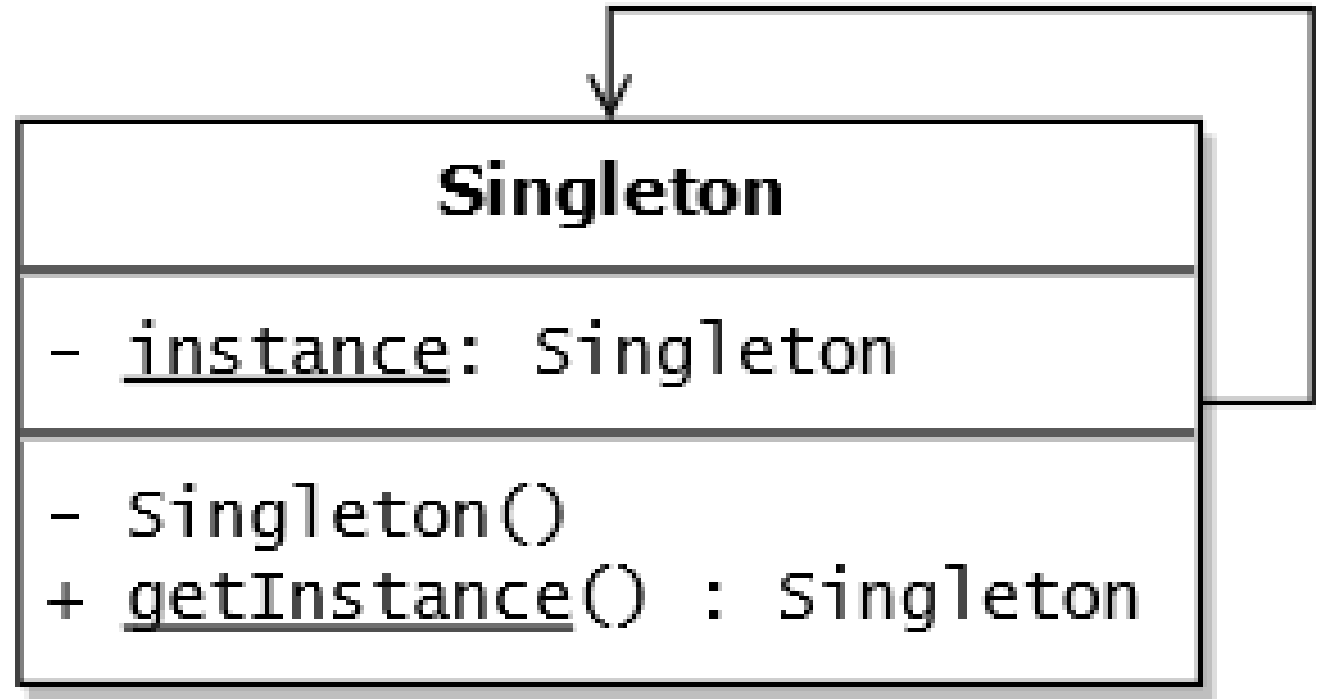
Singleton

Exemple 1:

```
1  class Singleton {
2      // attribut privé qui fait référence à l'objet
3      private static Singleton objet;
4
5      // constructeur de la classe Singleton
6      private Singleton() {
7          // corps
8      }
9
10     public static Singleton getInstance() {
11         // écrire du code qui nous permet de créer un seul objet,
12         // accéder à l'objet selon nos besoins
13
14         // créer un objet s'il n'est pas déjà créé
15         if (objet == null) {
16             objet = new Singleton();
17         }
18         // renvoie l'objet singleton
19         return objet;
20
21     }
22 }
23
```

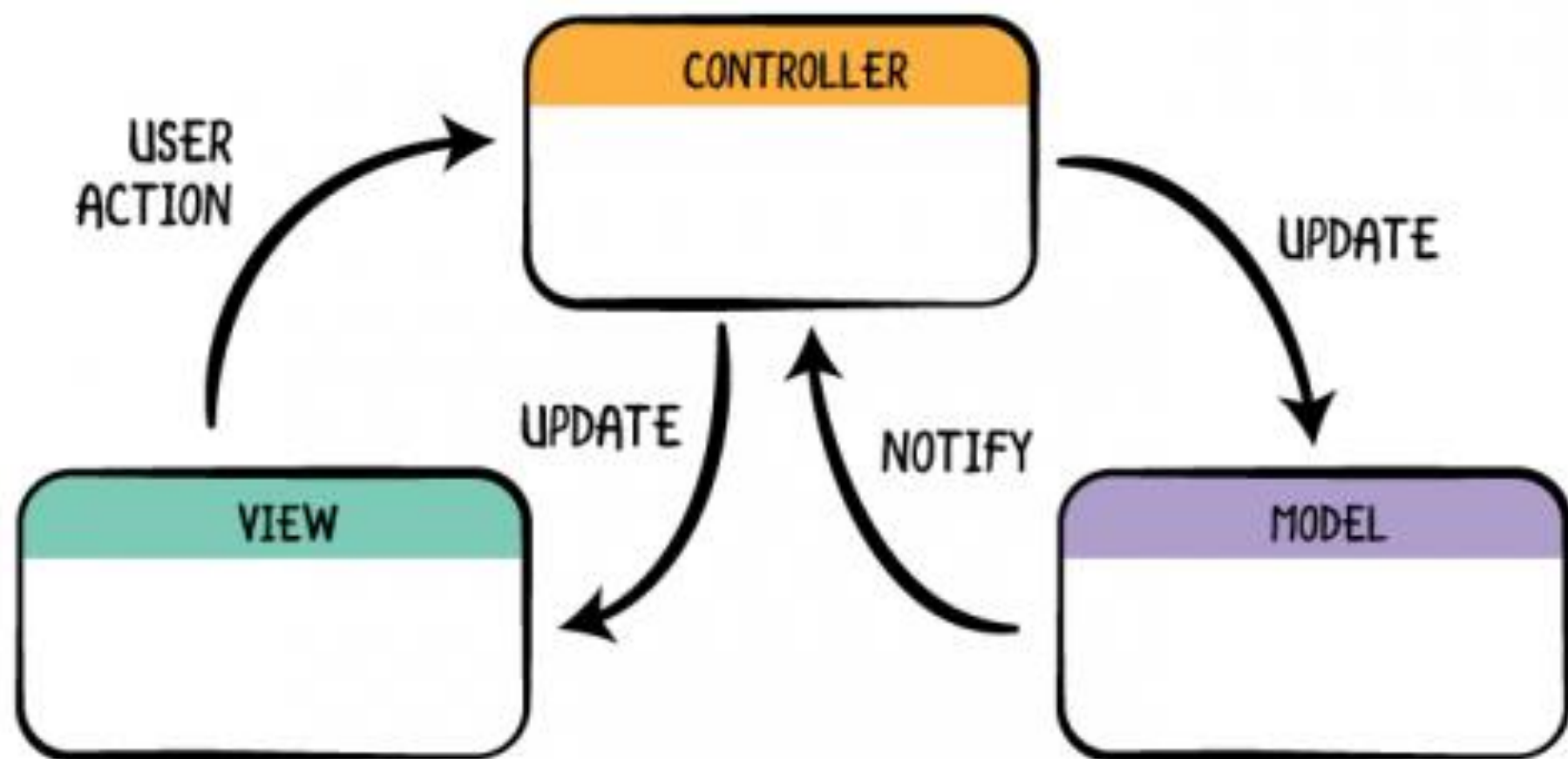


Singleton

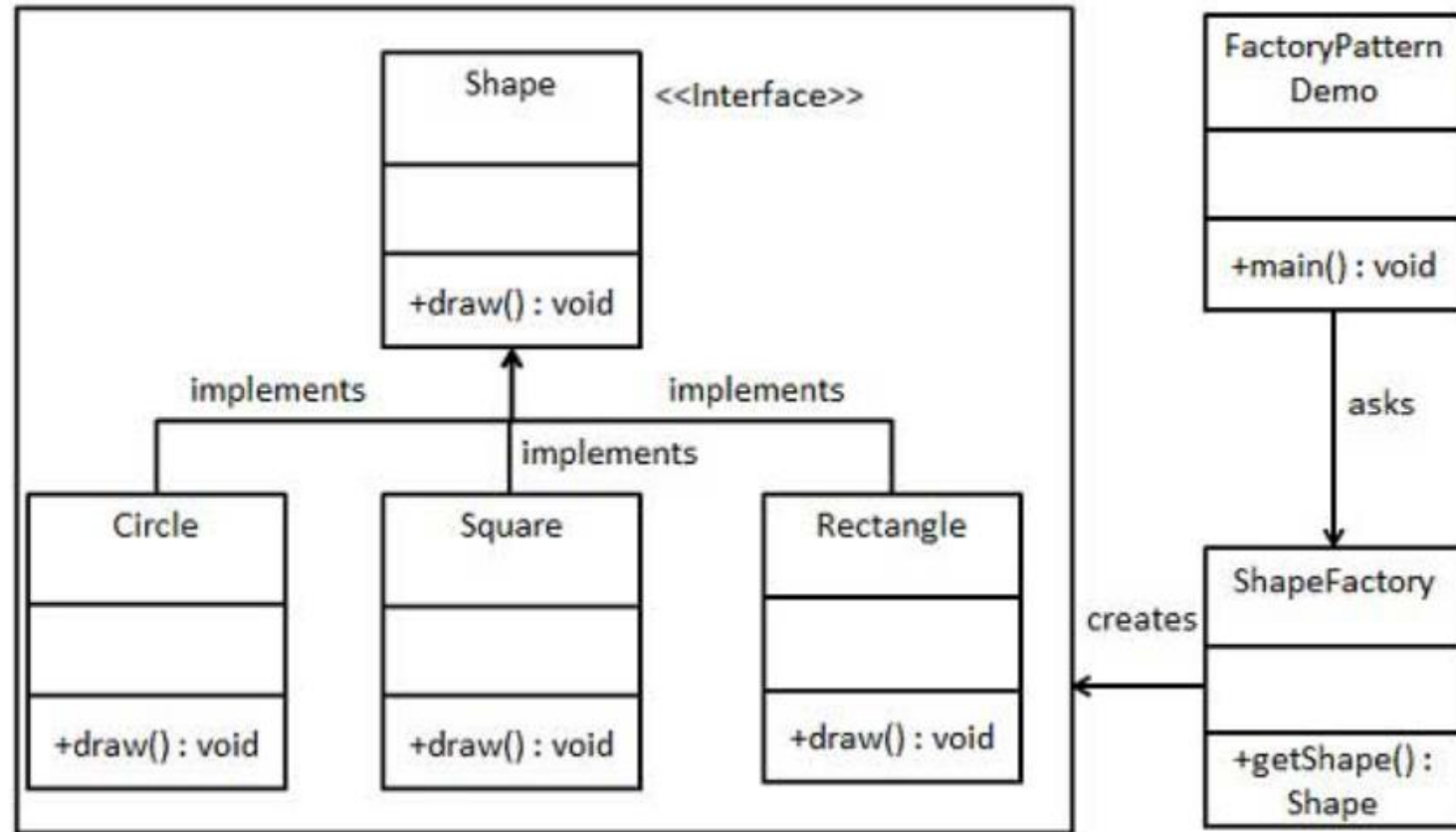




Achievement unlocked



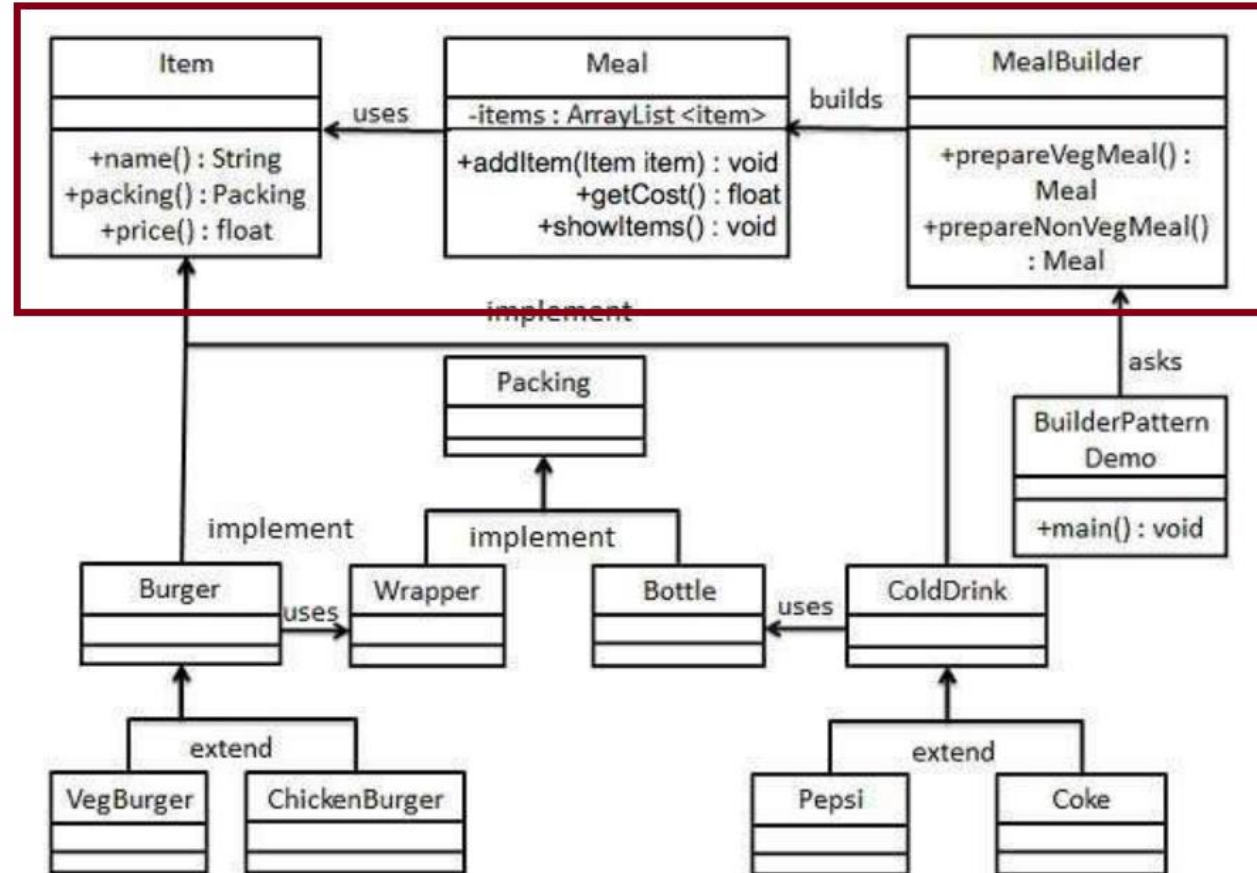
Pattern Factory



Pattern Factory

```
public class ShapeFactory {  
  
    //use getShape method to get object of type shape  
    public Shape getShape(String shapeType){  
        if(shapeType == null){  
            return null;  
        }  
        if(shapeType.equalsIgnoreCase("CIRCLE")){  
            return new Circle();  
        } else if(shapeType.equalsIgnoreCase("RECTANGLE")){  
            return new Rectangle();  
        } else if(shapeType.equalsIgnoreCase("SQUARE")){  
            return new Square();  
        }  
  
        return null;  
    }  
}
```

Pattern Builder



Pattern Builder

```
public class MealBuilder {  
  
    public Meal prepareVegMeal () {  
        Meal meal = new Meal();  
        meal.addItem(new VegBurger());  
        meal.addItem(new Coke());  
        return meal;  
    }  
  
    public Meal prepareNonVegMeal () {  
        Meal meal = new Meal();  
        meal.addItem(new ChickenBurger());  
        meal.addItem(new Pepsi());  
        return meal;  
    }  
}
```