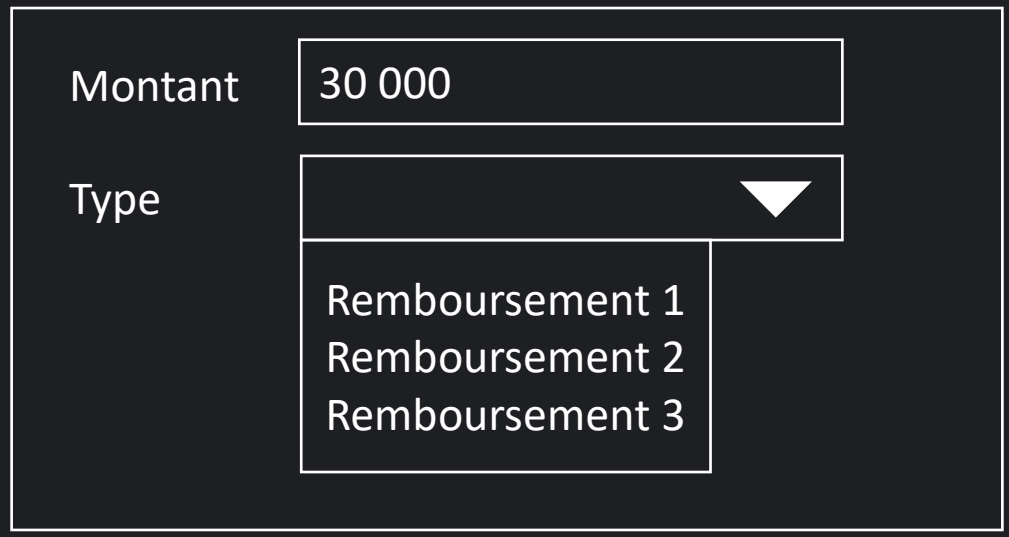
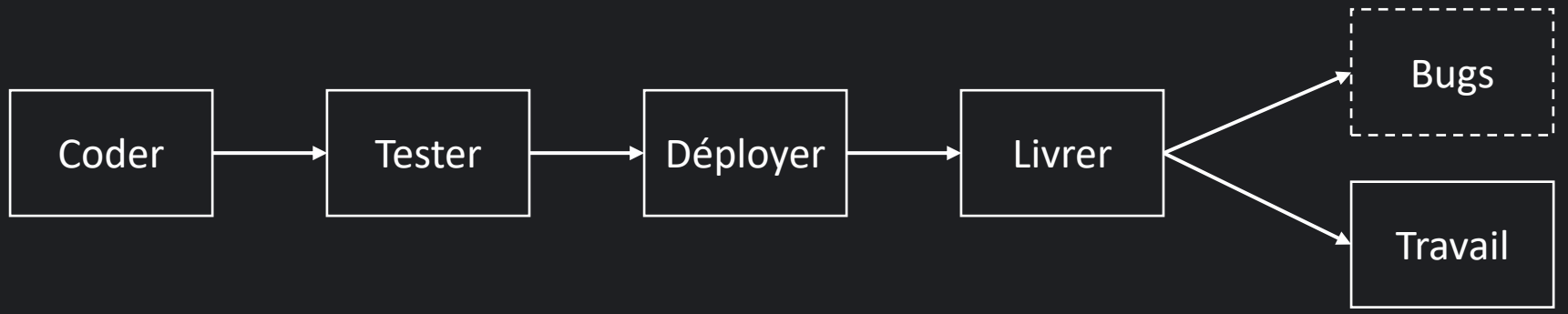
Patron stratégie

Patrons de conception

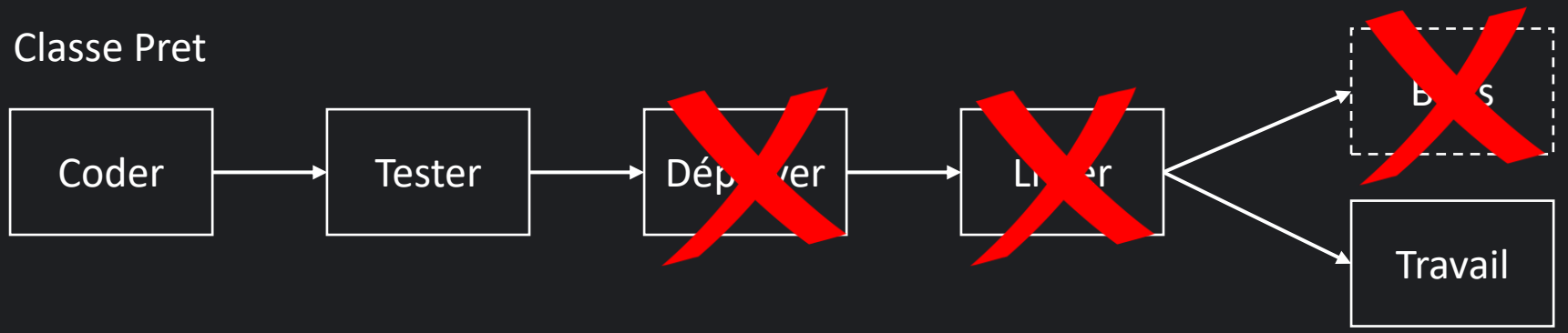
Le problème


Montant
30 000
Type
Remboursement 1
Remboursement 2
Remboursement 3

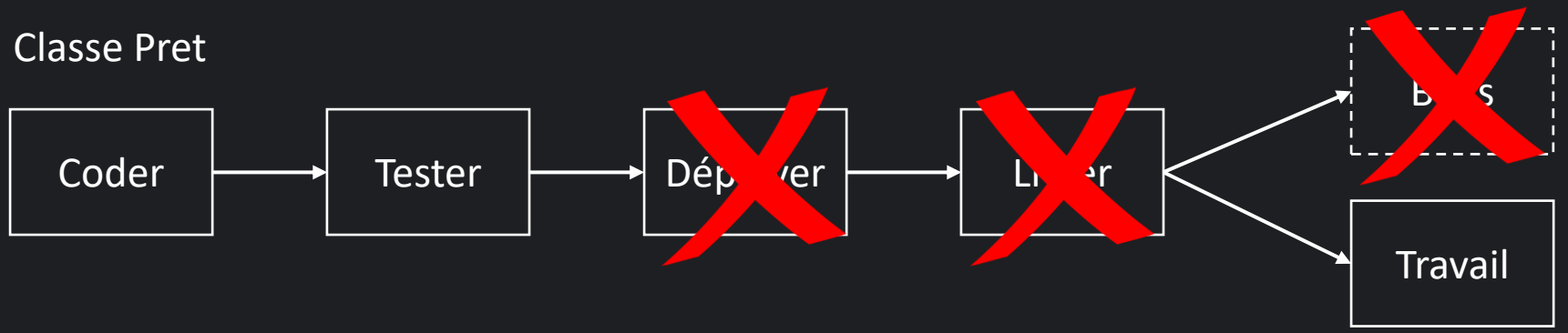
```
public class Pret {
    public void simuler (int montant, String typeRemboursement) {
        if (typeRemboursement. equals ("Remboursement1")) {
        } else if (typeRemboursement. equals ("Remboursement2")) {
        } else if (typeRemboursement. equals ("Remboursement3")) {

        } else if(typeRemboursement.equals("Remboursement4")) {
            /* nouvelle stratégie ajoutée */

        } else {
            /* rien */
        }
    }
}
```



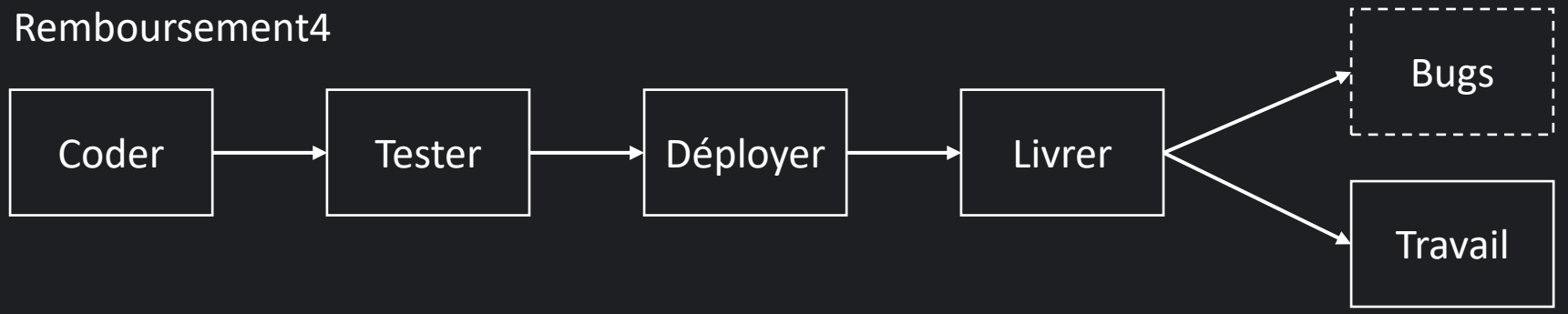
Classe Pret



Classe Pret



Remboursement4



Je veux

Ouvert à l'extension et fermé à la modification

Principe Ouvert/Fermé

Patron stratégie

Définition et mise en place

Définir chaque algorithme dans sa propre classe

Principe Stratégie

```
public void simuler (int montant, String typeRemboursement) {
    if (typeRemboursement.equals ("Remboursement1")) {
    } else if (typeRemboursement.equals ("Remboursement2")) {
    } else if (typeRemboursement.equals ("Remboursement3")) {
    } else {
        /* rien */
    }
}
```

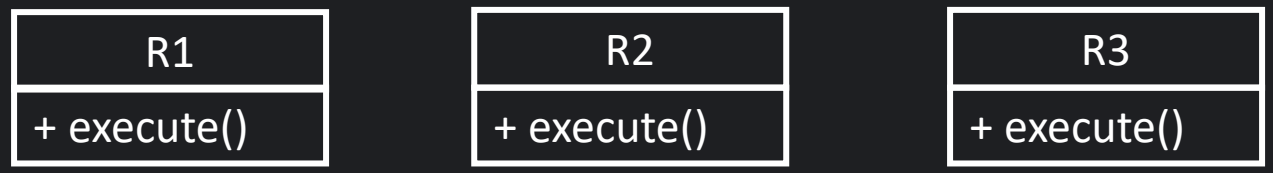
3 algorithmes : Remboursement1, Remboursement2, Remboursement3

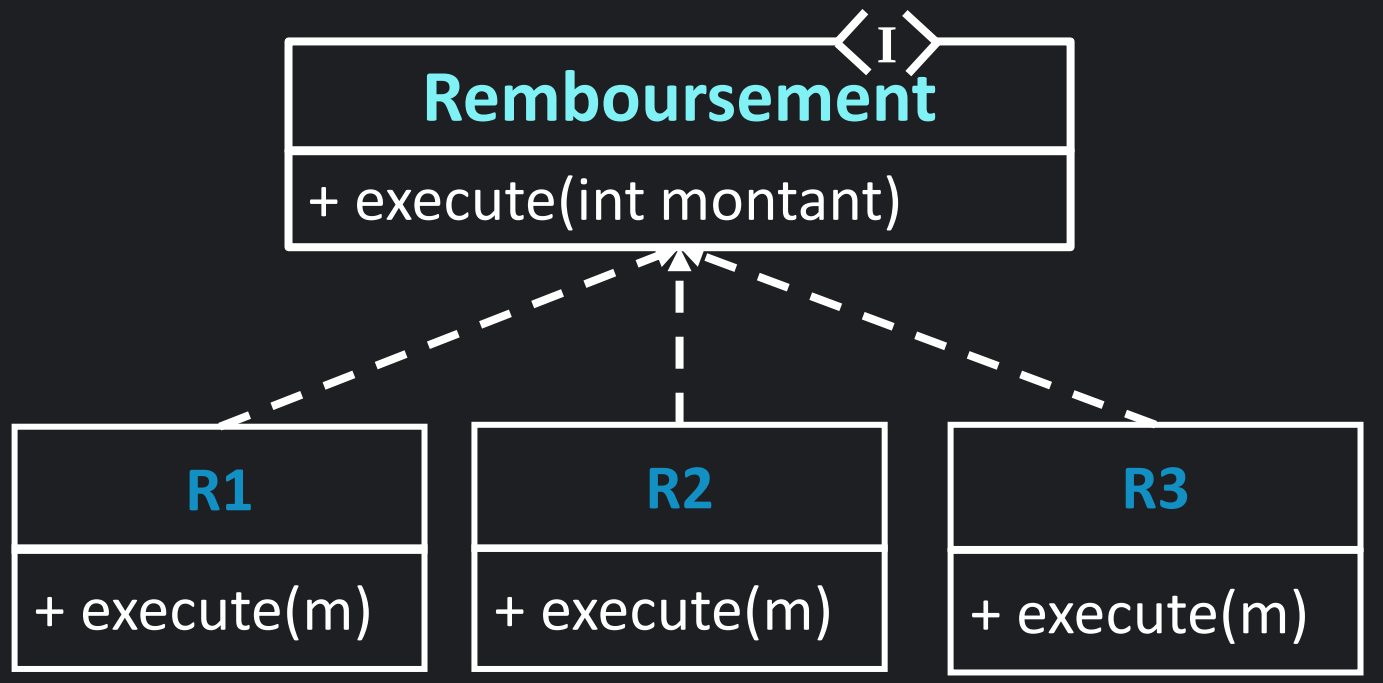
```
public    void   simuler  ( int    montant ,   String    typeRemboursement)  {
      if  ( typeRemboursement. equals ( "Remboursement1" ))   {
      }  else   if  ( typeRemboursement. equals ( " Remboursement2" ))   {
      }  else   if  ( typeRemboursement. equals ( " Remboursement3" ))   {
      }  else   {
           /*   rien */
      }
}
```

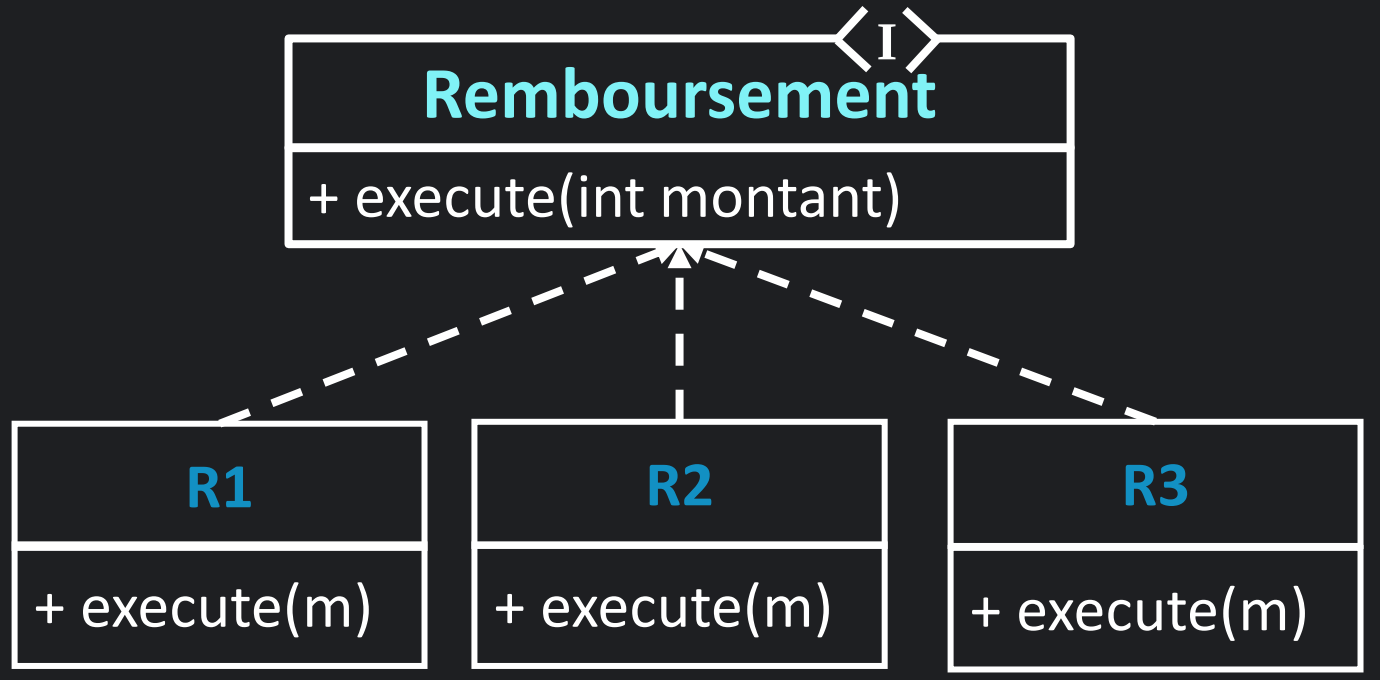
3 algorithmes : Remboursement1, Remboursement2, Remboursement3

```
public    void   simuler  ( int    montant ,   String    typeRemboursement)  {
      if  ( typeRemboursement. equals ( "Remboursement1" ))   {
      }  else   if  ( typeRemboursement. equals ( " Remboursement2" ))   {
      }  else   if  ( typeRemboursement. equals ( " Remboursement3" ))   {
      }  else   {
           /*   rien */
      }
}
```

3 algorithmes : Remboursement1, Remboursement2, Remboursement3

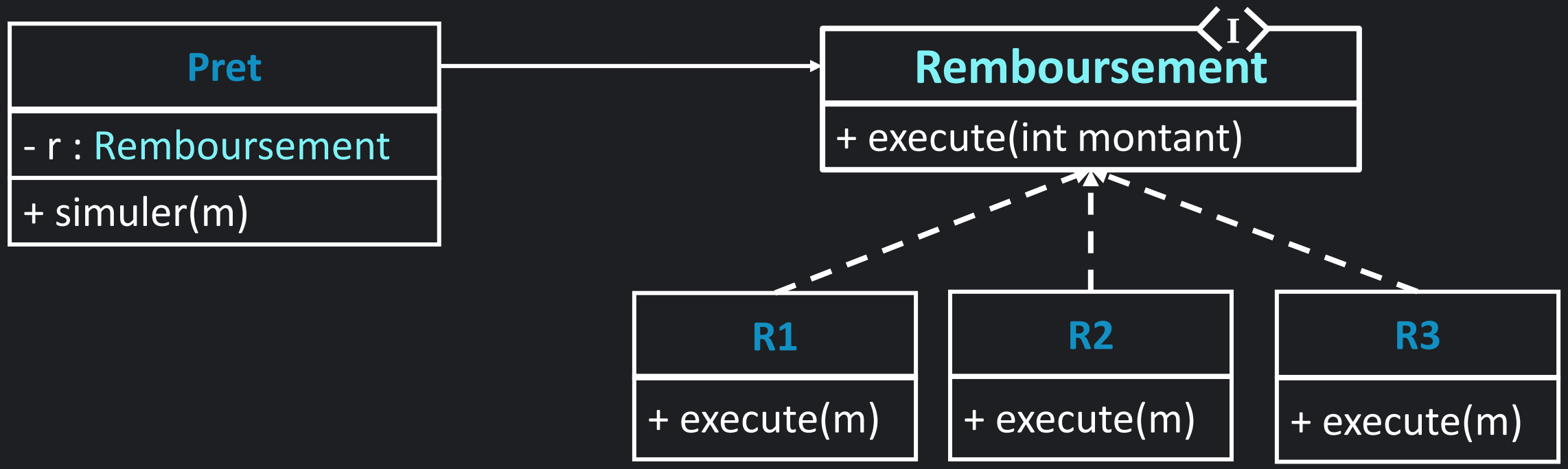



I
Remboursement
+ execute(int montant)
R1
+ execute(m)
R2
+ execute(m)
R3
+ execute(m)

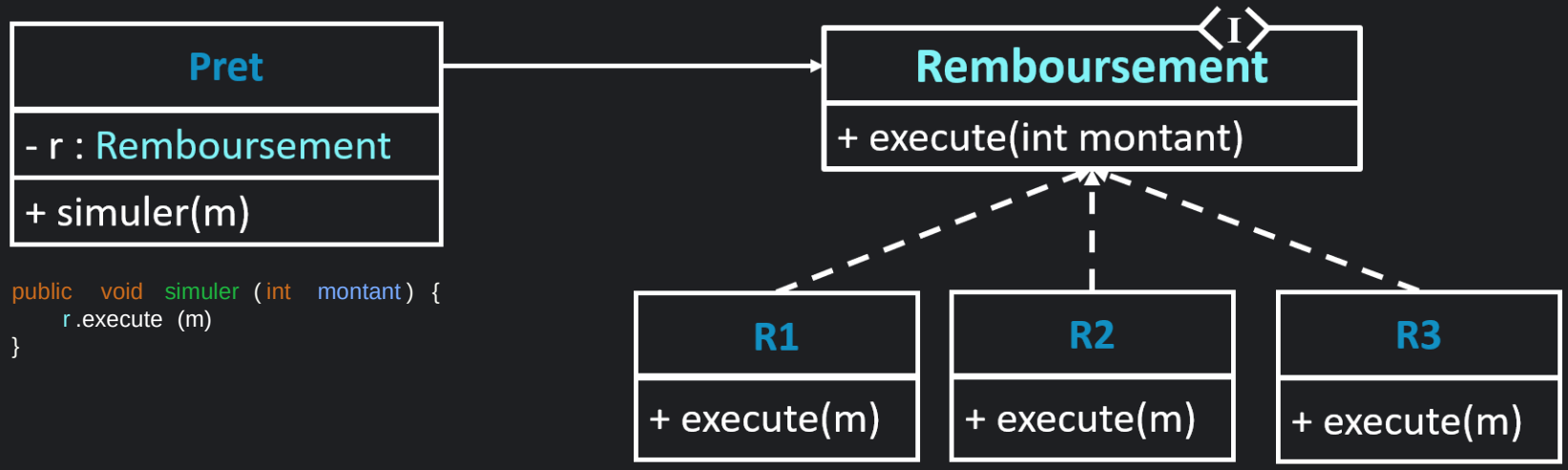



```java
public class R1 implements Remboursement {
    @Override
    public void execute (int montant) {
        System.out.println("R1");
    }
}
```

```java
public class R2 implements Remboursement {
    @Override
    public void execute (int montant) {
        System.out.println("R2");
    }
}
```



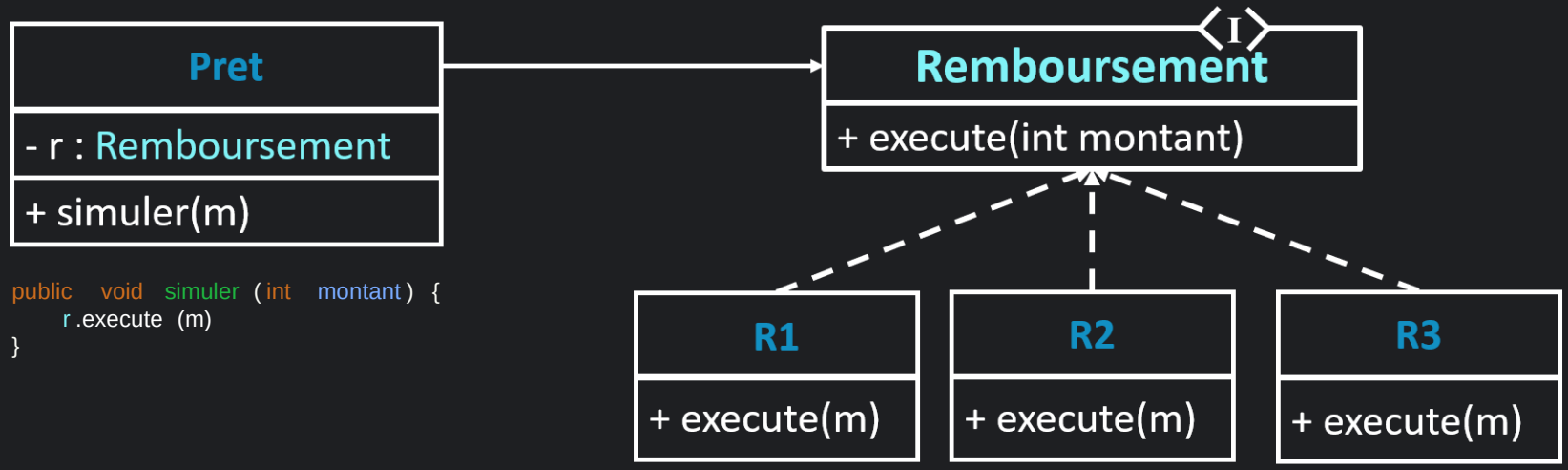

```
public    void   simuler  ( int    montant )  {
      r .execute  (m)
}

public    void   setRemboursement( Remsoursement r )  {
      this  . remboursement  =  r ;
}
```




```
public void simuler (int montant) {
    r.execute (m)
}
```

```
public static void main(String [] args) {
    Pret pret = new Pret () ;

    pret . setRemboursement( new Remboursement1()) ;
    pret . simuler (30000) ;
```

- r est de type Remboursement1
- appel de execute(30000) sur r



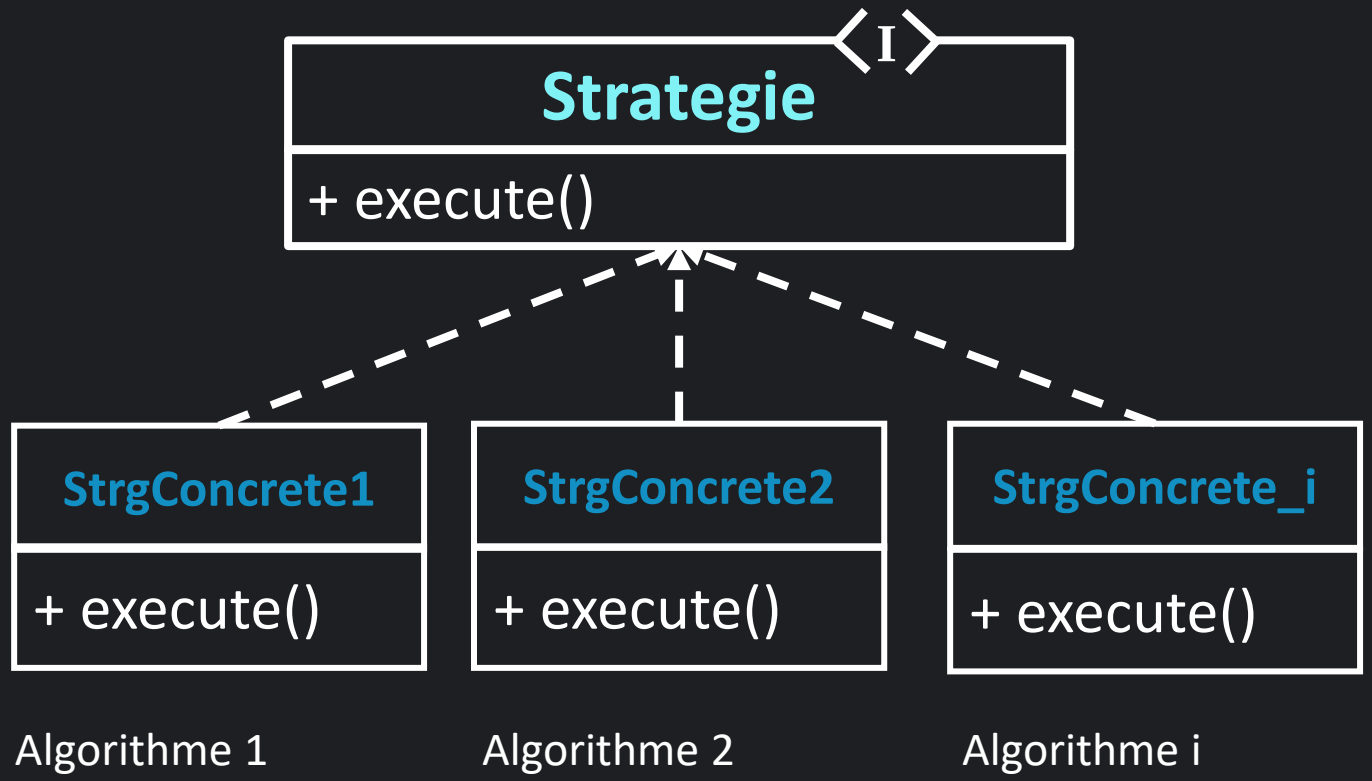

```
public void simuler (int montant) {
    r.execute (m)
}
```

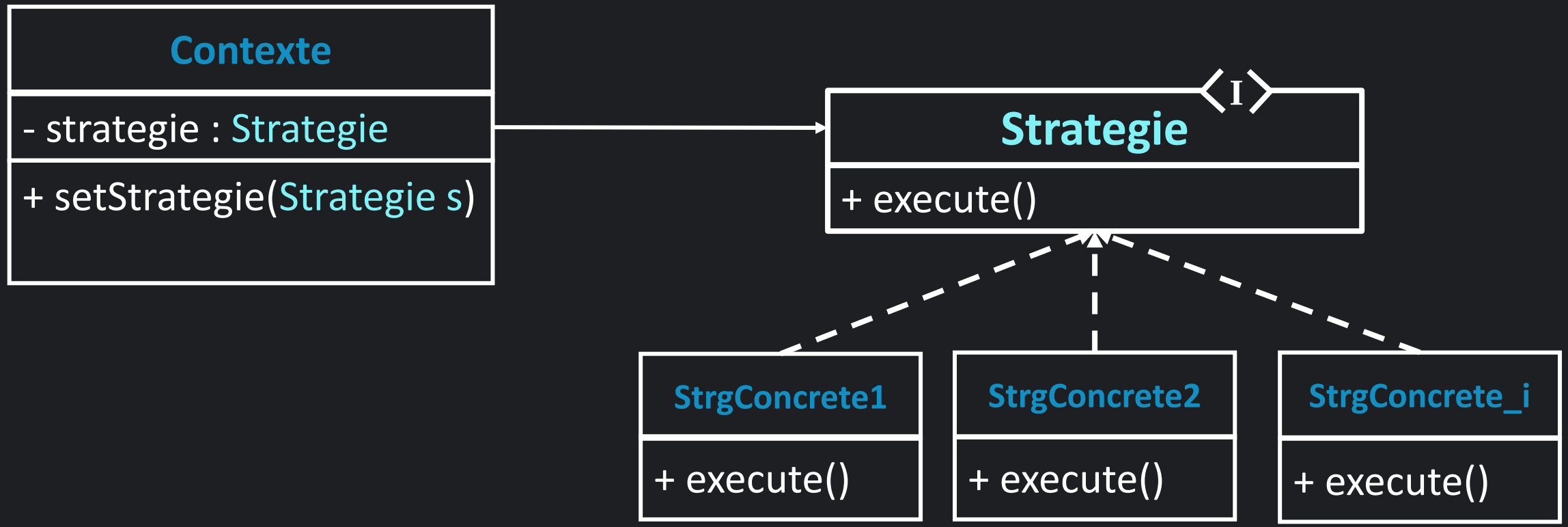
```
public static void main(String [] args) {
    Pret pret = new Pret () ;

    pret . setRemboursement( new Remboursement1()) ;
    pret . simuler (30000) ;

    pret . setRemboursement( new Remboursement2()) ;
    pret . simuler (30000) ;
```

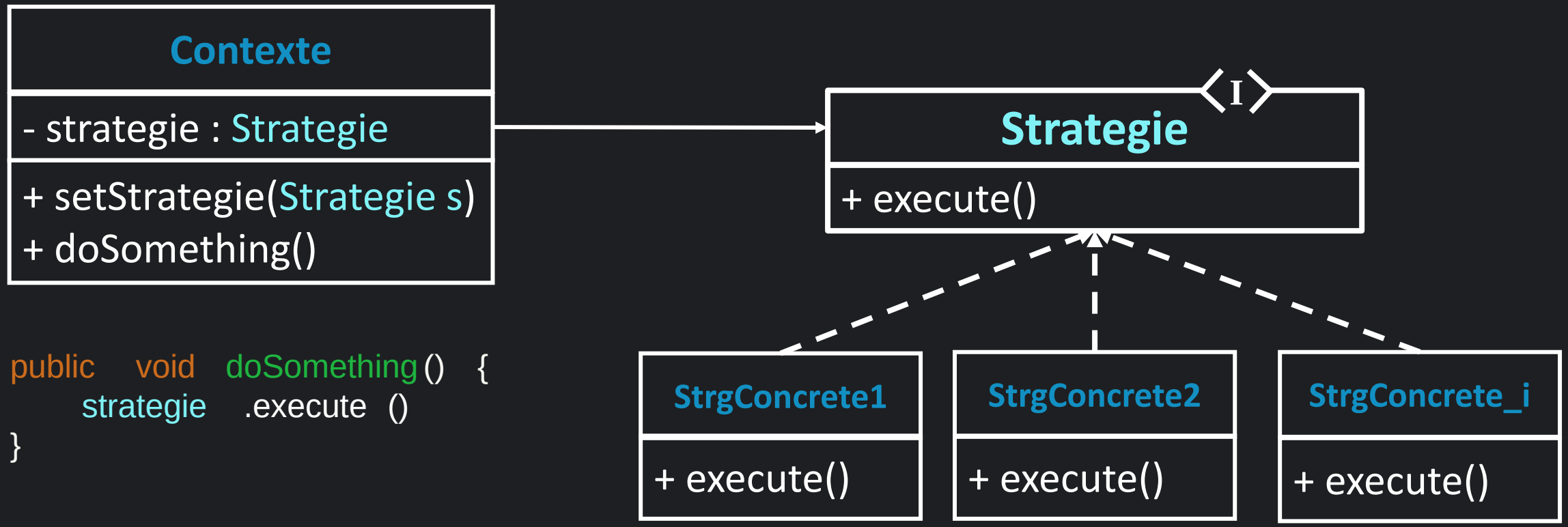
r est de type Remboursement2
appel de execute(30000) sur r

Comment ?

I
Strategie
+ execute()


I
Strategie
+ execute()
StrgConcrete1
+ execute()
StrgConcrete2
+ execute()
StrgConcrete_i
+ execute()
Algorithme 1
Algorithme 2
Algorithme i


Contexte
- strategie : Strategie
+ setStrategie(Strategie s)
I
Strategie
+ execute()
StrgConcrete1
+ execute()
StrgConcrete2
+ execute()
StrgConcrete_i
+ execute()




```
public    void   doSomething ()   {
      strategie    .execute   ()
}
```

Quand ?

Quand on a différentes variantes d’un algorithme et qu’on souhaite pouvoir passer d’un algorithme à un autre à l’éxécution

Quand on a différentes variantes d'un algorithme et qu'on souhaite pouvoir passer d'un algorithme à un autre à l'éxécution

Si une classe possède un gros bloc conditionnel qui choisit entre plusieurs algorithmes

Quand on a différentes variantes d'un algorithme et qu'on souhaite pouvoir passer d'un algorithme à un autre à l'éxécution

Si une classe possède un gros bloc conditionnel qui choisit entre plusieurs algorithmes

Séparer la logique métier (contexte) des divers algorithmes (les stratégies)

Quand on a différentes variantes d'un algorithme et qu'on souhaite pouvoir passer d'un algorithme à un autre à l'éxécution

Si une classe possède un gros bloc conditionnel qui choisit entre plusieurs algorithmes

Séparer la logique métier (contexte) des divers algorithmes (les stratégies)

Mise en œuvre

Sous Eclipse IDE

Mise en œuvre

Sous Eclipse IDE