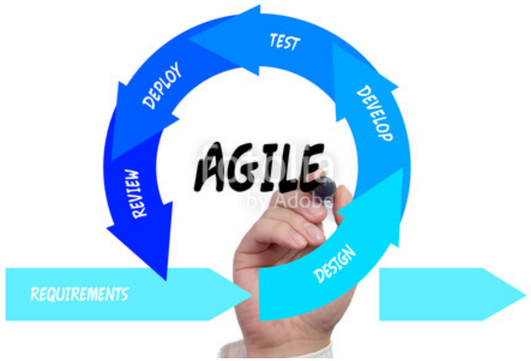


Classe Teoria Setmana 14: Disseny: Patrons de Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2021/22

UNIVERSITAT DE BARCELONA

Patró Composite

Nom del patró: **Composite**

Context: Creació d'objectes

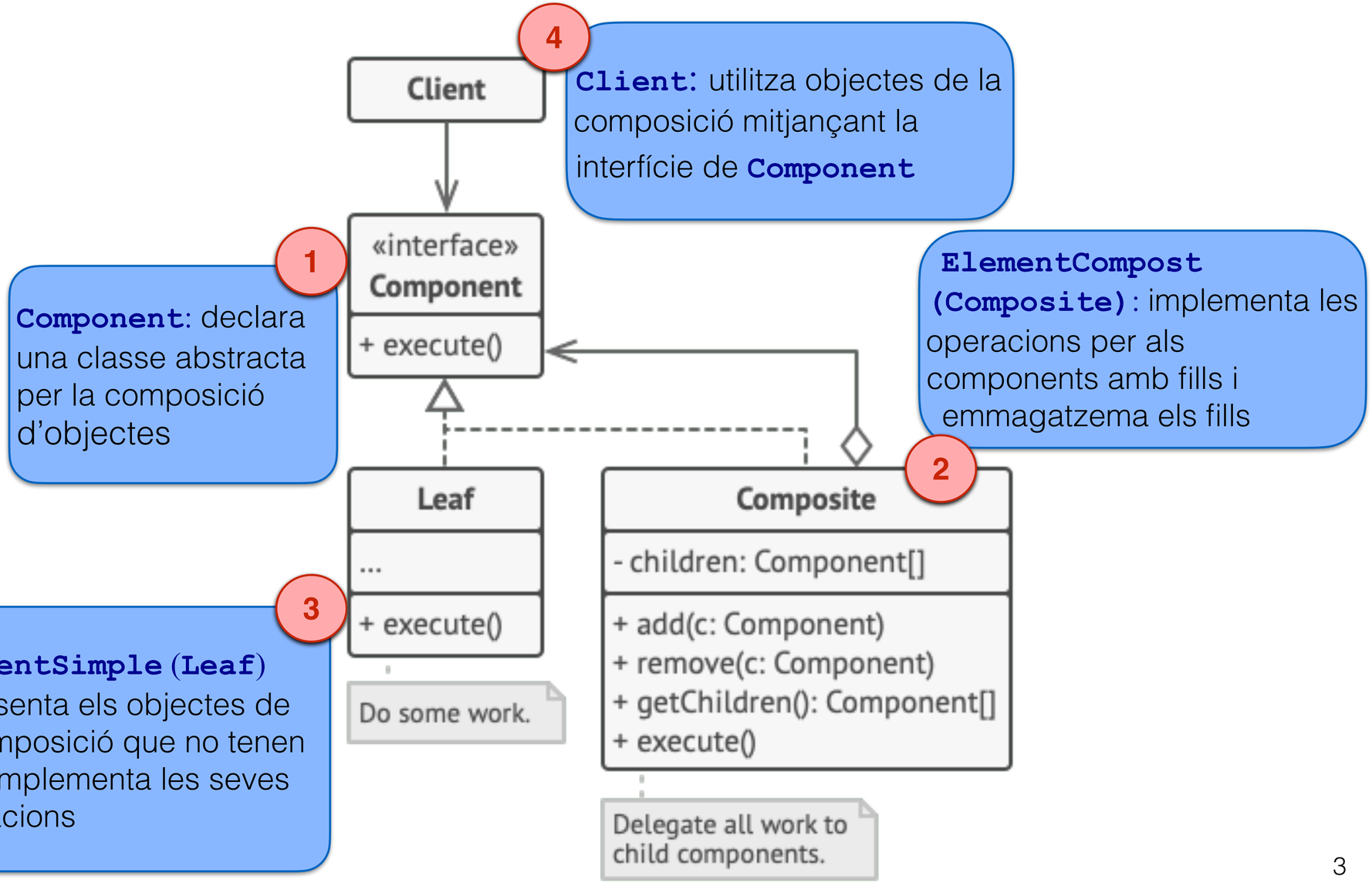
Problema:

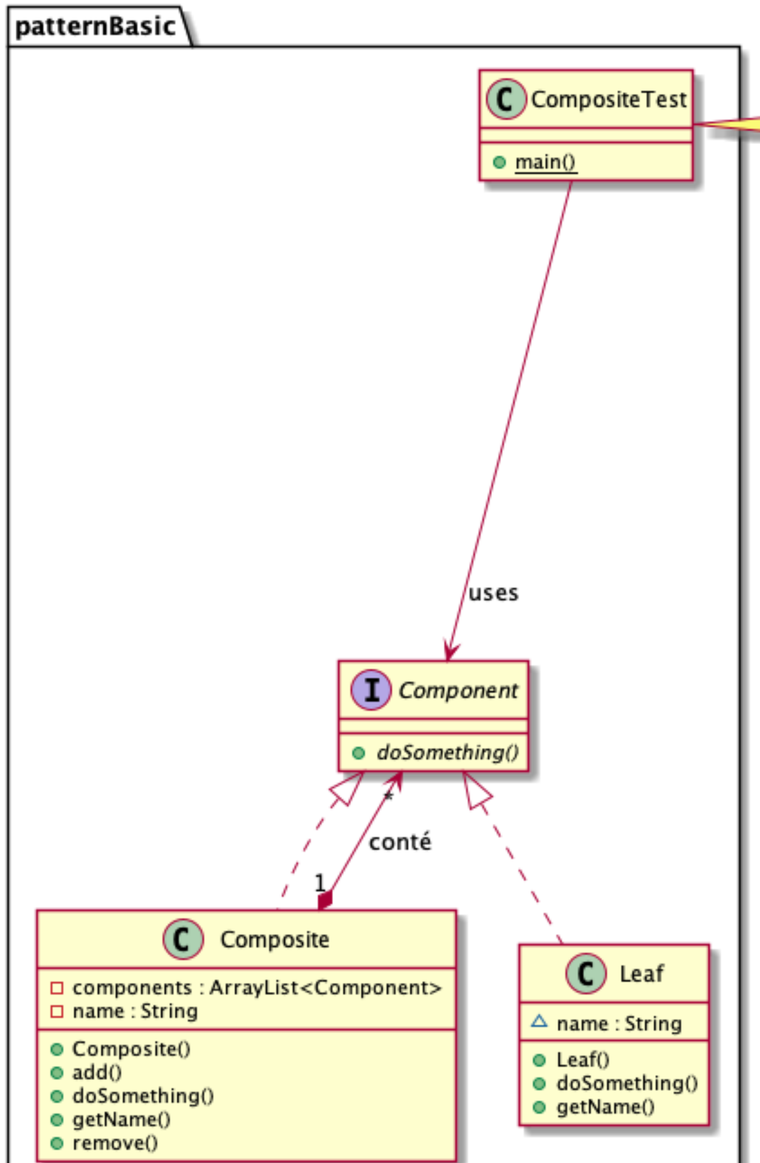
- Composar objectes en jerarquies **TOT-PART** i permetre als clients tractar objectes simples i compostos de manera uniforme

Solució:

- Defineix jerarquies de classes que tenen objectes primitius i objectes compostos a la vegada que els compostos estan formats per objectes primitius o altres objectes compostos.
- Utilització de l'herència per modelar els diferents tipus d'objectes (simples i compostos)
- Utilització de la composició per modelar les relacions TOT-PART dels compostos

Patró Composite

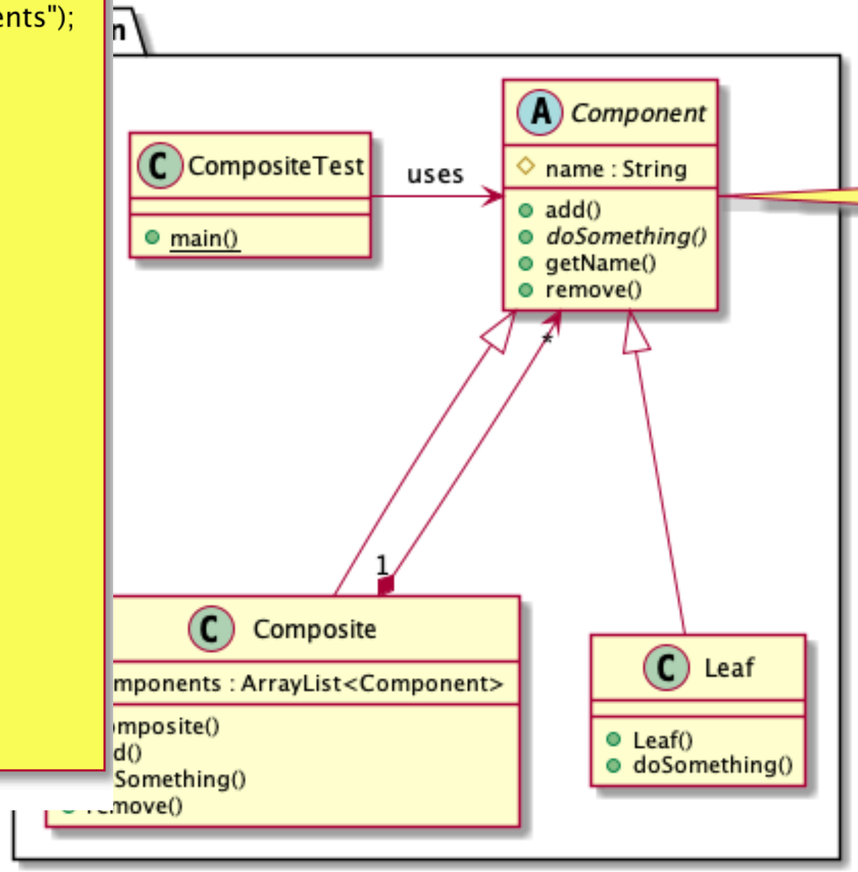





```
public static void main(String[] args) {

        Component componentOne = new Composite("Composite One");
        Component componentTwo = new Composite("Composite Two");
        Component componentThree = new Composite("Composite Three");

        Component componentWrapper = new Composite("All components");

        ((Composite) componentWrapper).add(componentOne);
        ((Composite) componentWrapper).add(componentTwo);
        ((Composite) componentWrapper).add(componentThree);

        ((Composite) componentOne).add(new Leaf("ONE: Sub component one"));
        ((Composite) componentOne).add(new Leaf("ONE: Sub component two"));
        ((Composite) componentOne).add(new Leaf("ONE: Sub component three"));

        ((Composite) componentTwo).add(new Leaf("TWO: Sub component one"));
        ((Composite) componentTwo).add(new Leaf("TWO: Sub component two"));

        ((Composite) componentThree).add(new Leaf("THREE: Sub component one"));
        ((Composite) componentThree).add(new Leaf("THREE: Sub component two"));
        ((Composite) componentThree).add(new Leaf("THREE: Sub component three"));
        ((Composite) componentThree).add(new Leaf("THREE: Sub component four"));


        componentWrapper.doSomething();

    }
```

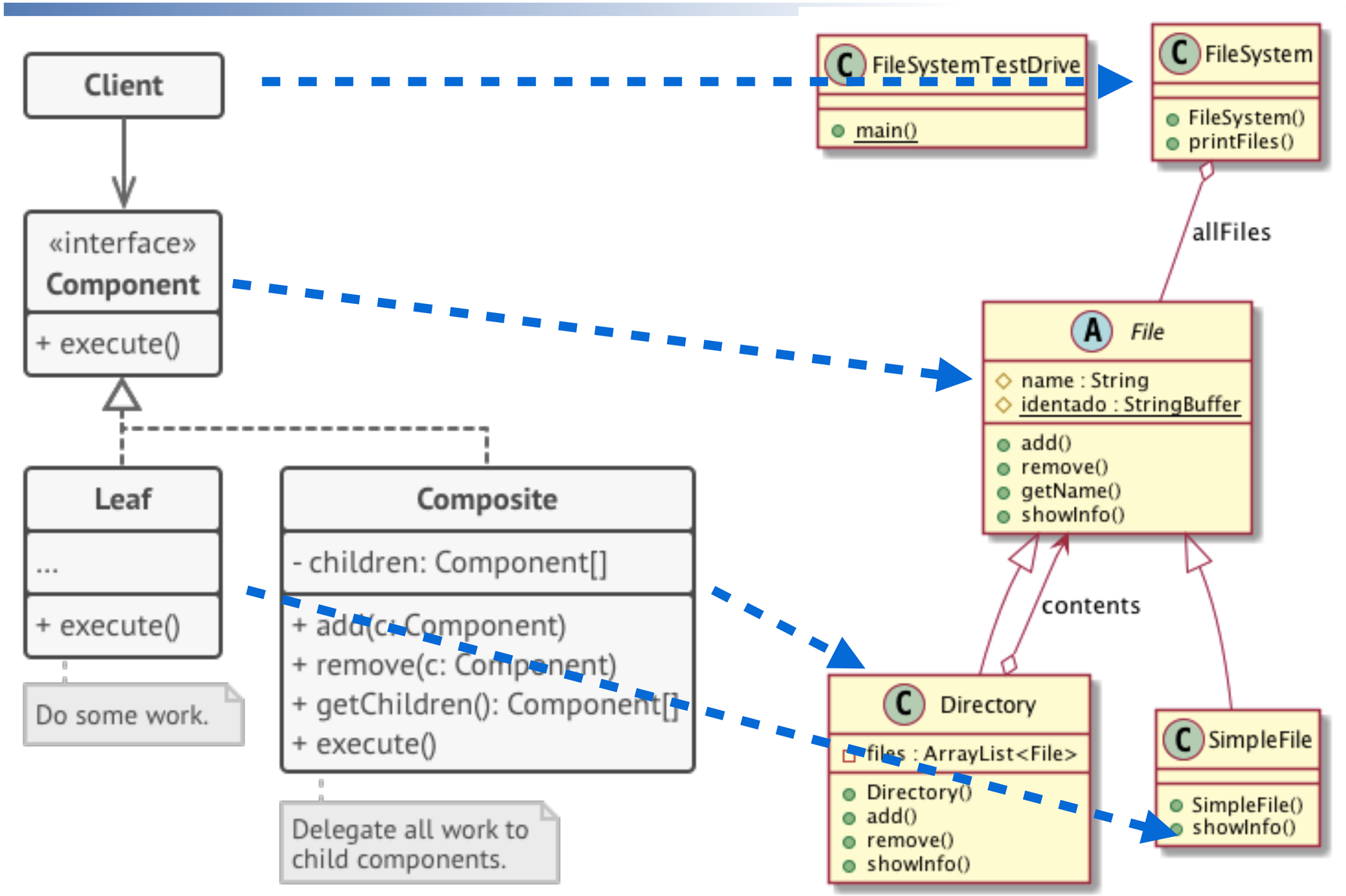
Canvi a abstract class

```
public class CompositeTest {

    public static void main(String[] args) {

        Component componentOne = new Composite("Composite One");
        Component componentTwo = new Composite("Composite Two");
        Component componentThree = new Composite("Composite Three");

        Component componentWrapper = new Composite("All components");

        componentWrapper.add(componentOne);
        componentWrapper.add(componentTwo);
        componentWrapper.add(componentThree);

        componentOne.add(new Leaf("ONE: Sub component one"));
        componentOne.add(new Leaf("ONE: Sub component two"));
        componentOne.add(new Leaf("ONE: Sub component three"));

        componentTwo.add(new Leaf("TWO: Sub component one"));
        componentTwo.add(new Leaf("TWO: Sub component two"));

        componentThree.add(new Leaf("THREE: Sub component one"));
        componentThree.add(new Leaf("THREE: Sub component two"));
        componentThree.add(new Leaf("THREE: Sub component three"));
        componentThree.add(new Leaf("THREE: Sub component four"));


        componentWrapper.doSomething();

    }
```




```
public abstract class Component {

    protected String name;

    public void add(Component component) {
        throw new UnsupportedOperationException();
    }
    public void remove(Component component) {
        throw new UnsupportedOperationException();
    }
    public abstract void doSomething();

    public String getName() {
        return name;
    }

}
```

Exercicis

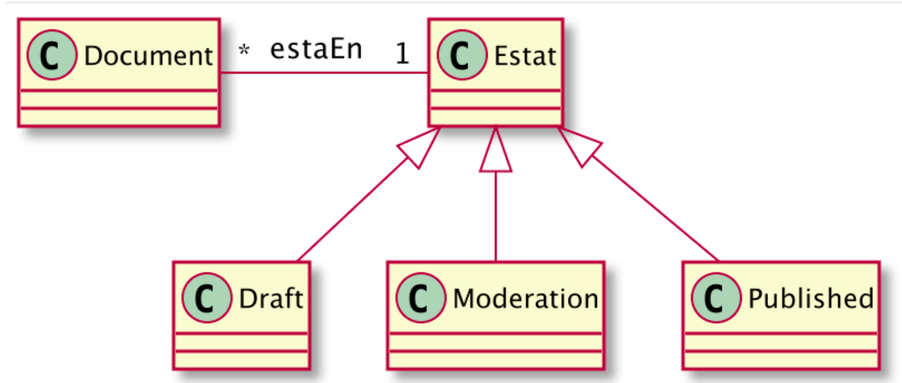

Client
«interface»
Component
+ execute()
Leaf
...
+ execute()
Do some work.
Composite
- children: Component[]
+ add(c: Component)
+ remove(c: Component)
+ getChildren(): Component[]
+ execute()
Delegate all work to child components.
FileSystemTestDrive
main()
FileSystem
FileSystem()
printFiles()
allFiles
File
name : String
identado : StringBuffer
add()
remove()
getName()
showInfo()
contents
Directory
files : ArrayList<File>
Directory()
add()
remove()
showInfo()
SimpleFile
SimpleFile()
showInfo()

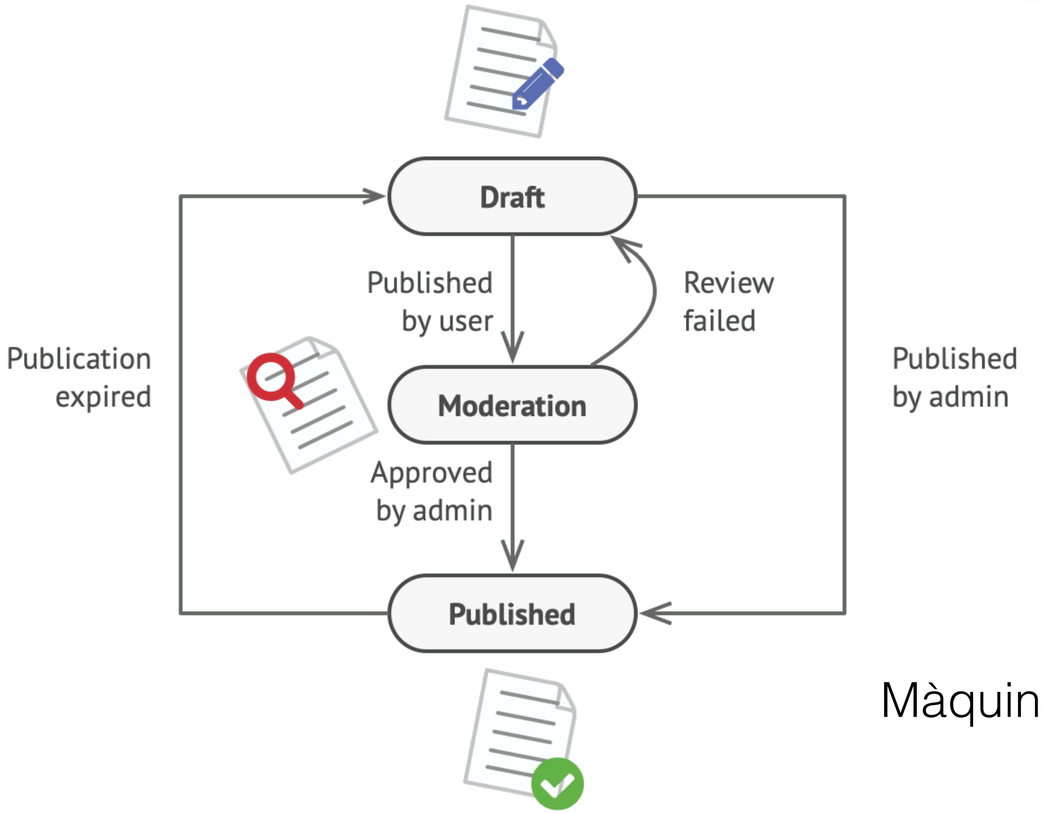
3.4. Patrons de disseny

Propòsit → **Àmbit ↓**	**CREACIÓ**	**ESTRUCTURA**	**COMPORTAMENT**
CLASSE	• Factory method	• class Adapter	• Interpreter • Template method
OBJECTE	• Abstract Factory • Builder • Prototype • Singleton • Object pool	• Object Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsability • Command • Iterator • Mediator • Memento • Observer • **State** • Strategy • Visitor

Exemple patró State

- Edició d'un document té 3 estats:
  Draft, Moderat, Publicat

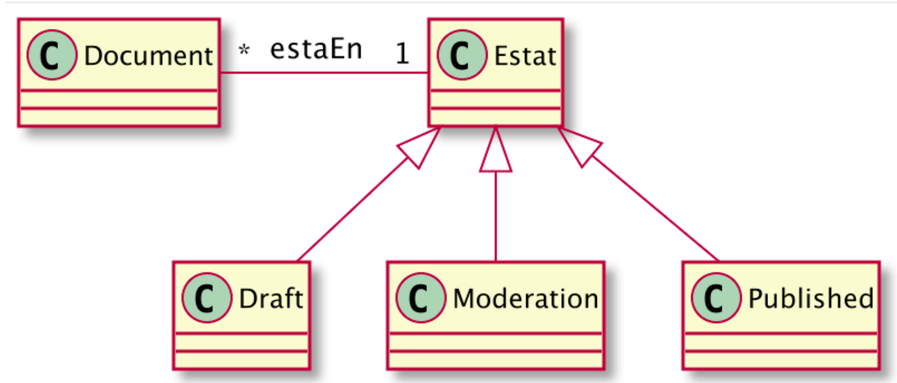


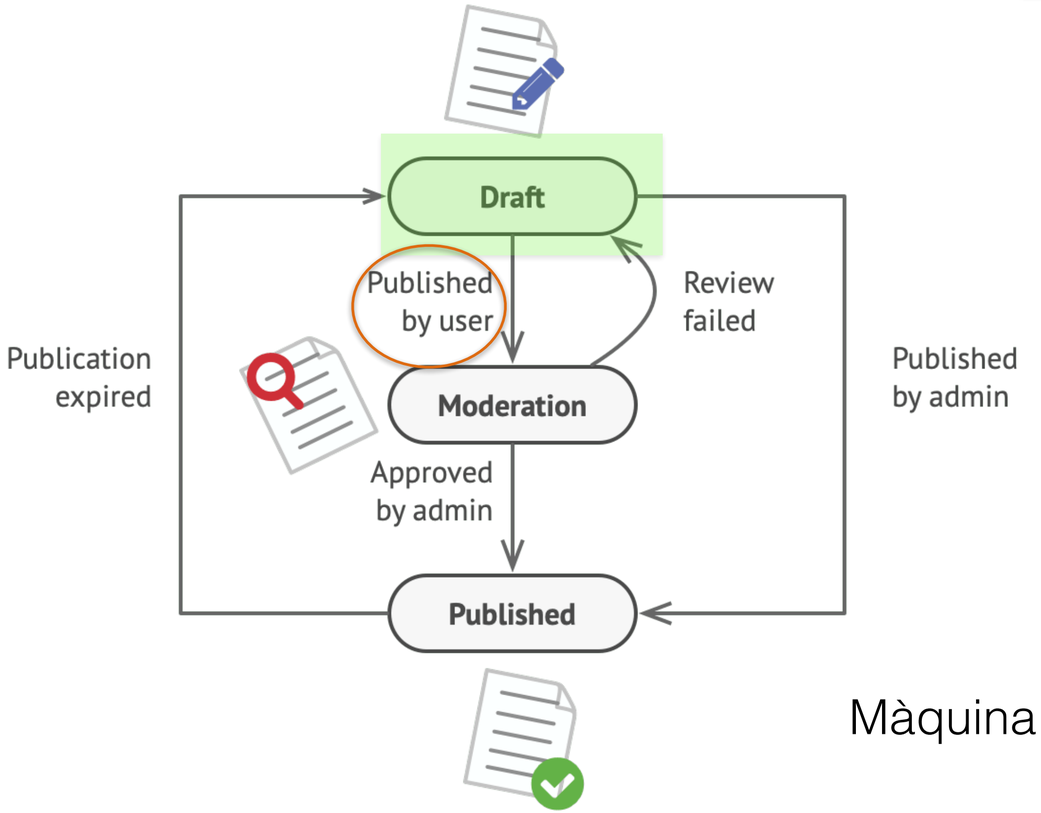


Màquina d'estats (State Machine)

Exemple patró State

- Edició d'un document té 3 estats:
  Draft, Moderat, Publicat

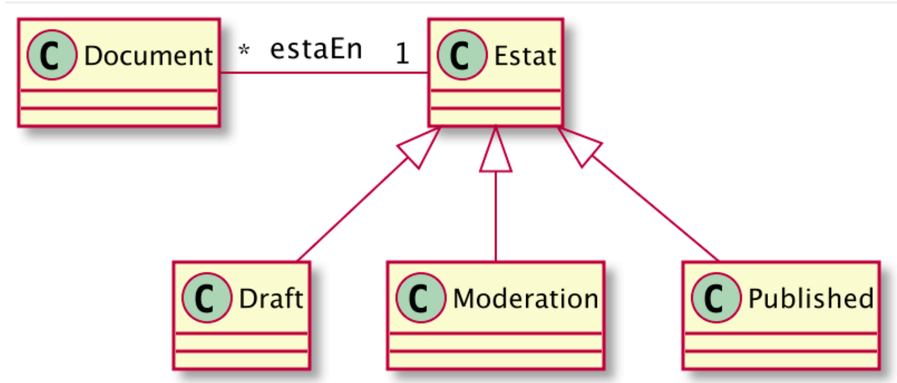


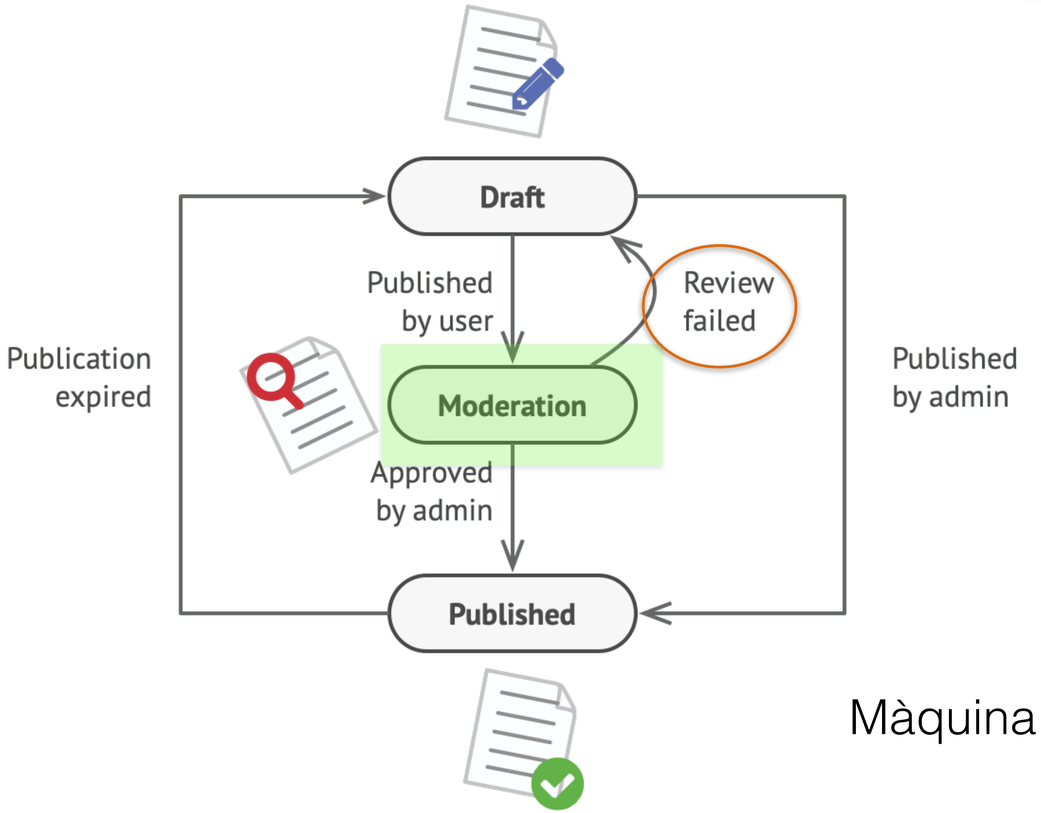


Màquina d'estats (State Machine)

Exemple patró State

- Edició d'un document té 3 estats:
  Draft, Moderat, Publicat

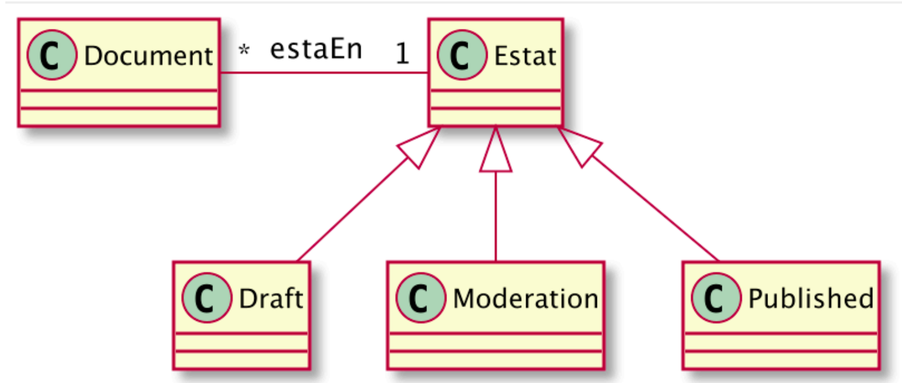


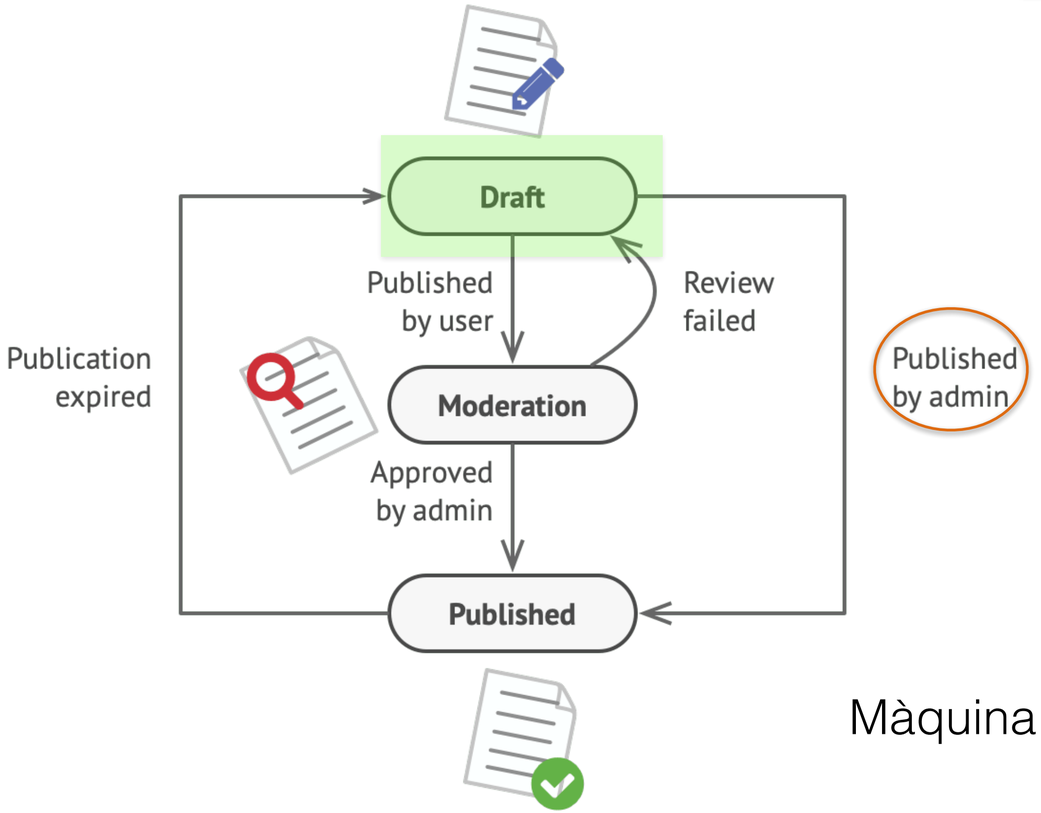


Màquina d'estats (State Machine)

Exemple patró State

- Edició d'un document té 3 estats:
  Draft, Moderat, Publicat

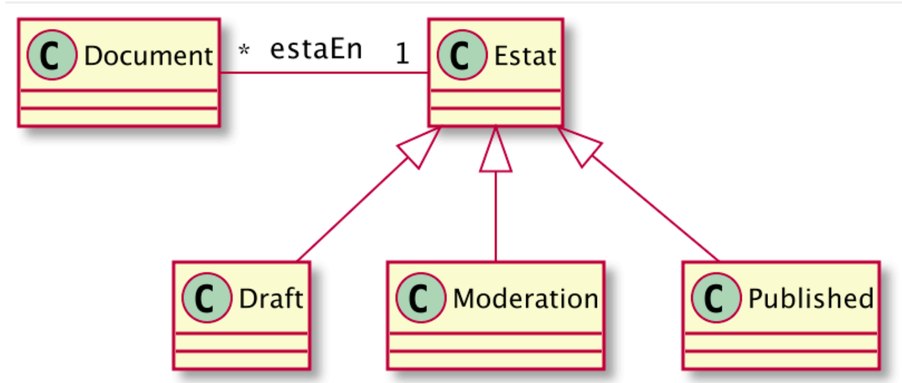


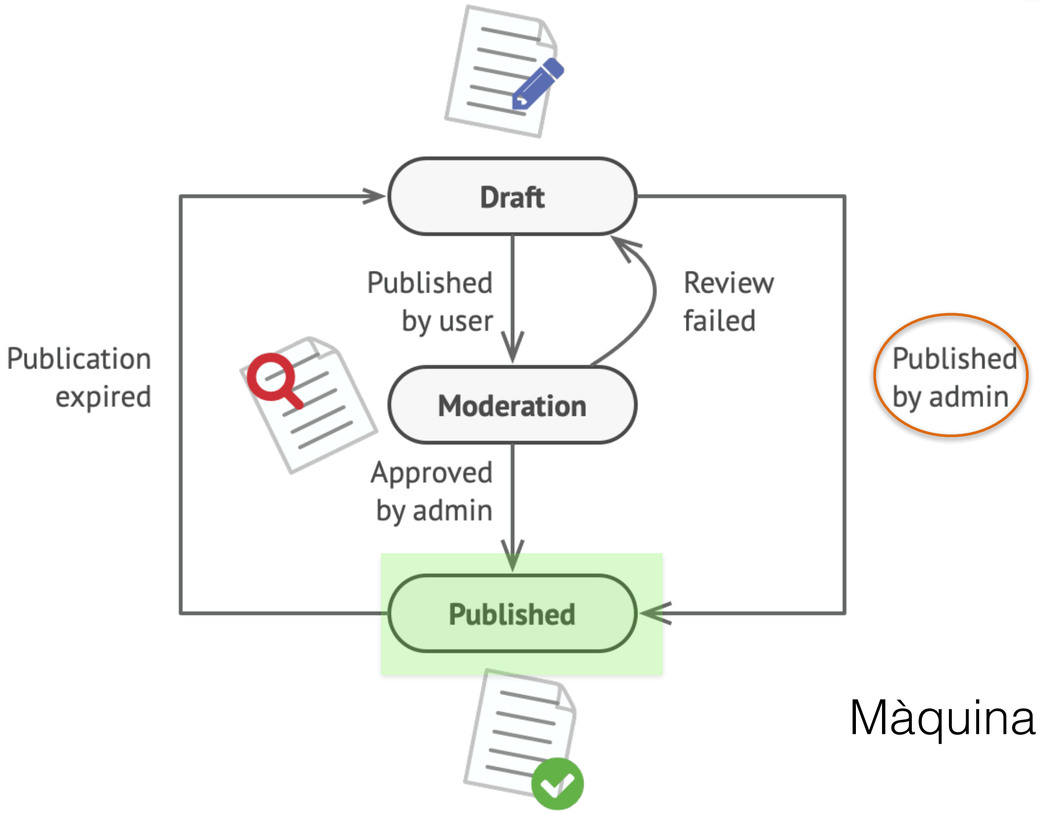


Màquina d'estats (State Machine)

Exemple patró State

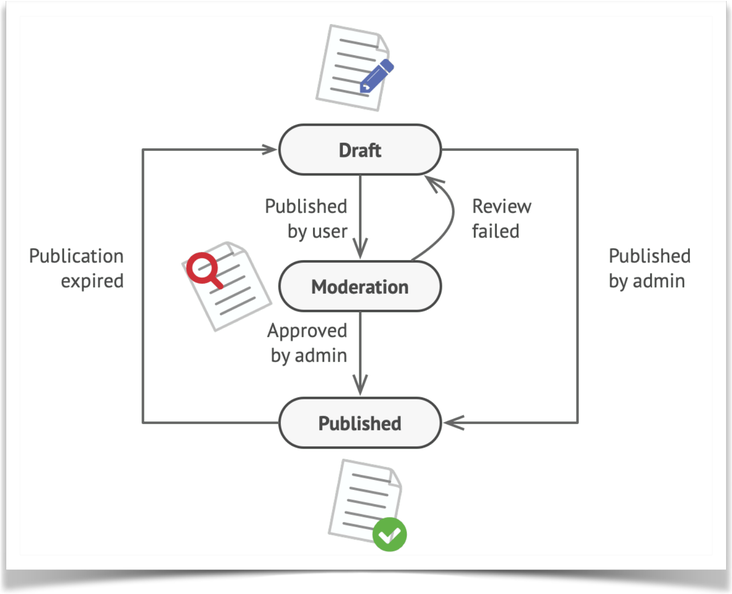
- Edició d'un document té 3 estats:
  Draft, Moderat, Publicat

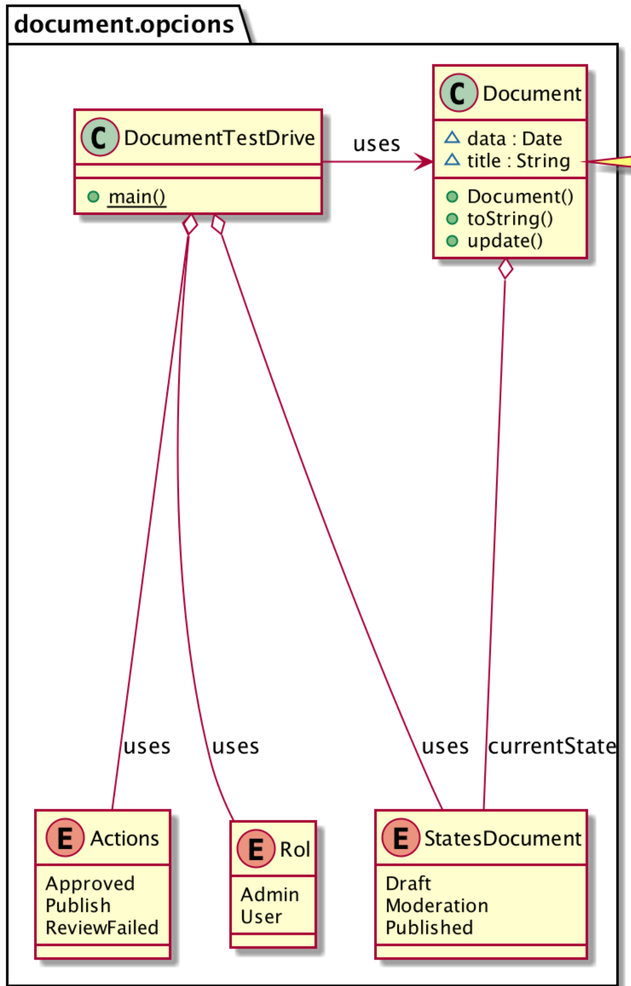




Màquina d'estats (State Machine)

Primera implementació





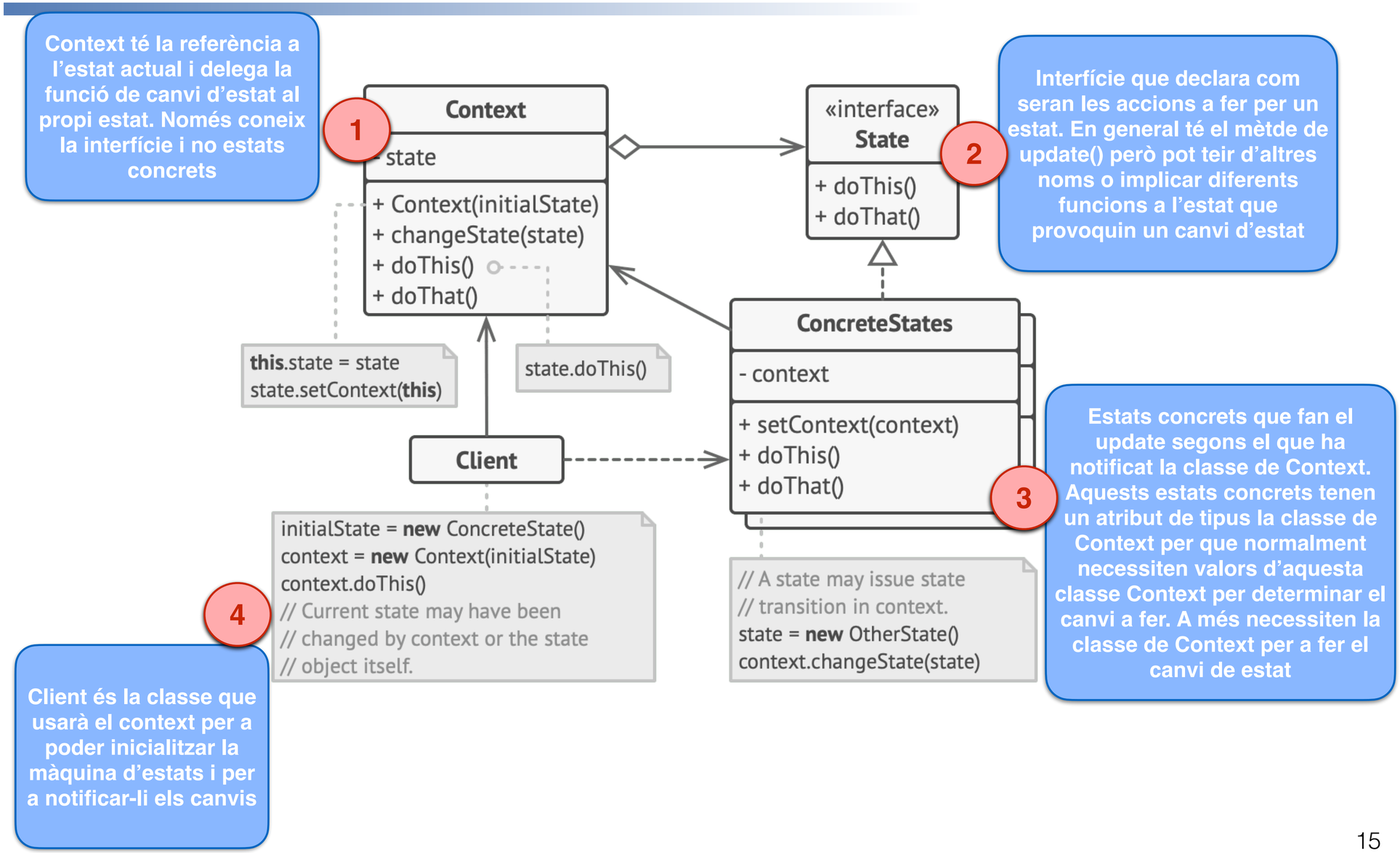

```
StatesDocument currentState;
Date data;
String title;

public Document(String title, Date data) {
   this.title = title;
   this.data = data;
   this.currentState = StatesDocument.Draft;
}

public void update(Rol role, Actions action ) {
   System.out.println("The role: "+role + " is doing "+action);
   switch (currentState) {
      case Draft:
         if (action == Actions.Publish) {
            if (role == Rol.User)
               currentState = StatesDocument.Moderation;
            else if (role == Rol.Admin)
               currentState = StatesDocument.Published;
         }
         break;
      case Moderation:
         if (role == Rol.Admin) {
            if (action == Actions.Approved)
               currentState = StatesDocument.Published;
            else if (action == Actions.ReviewFailed)
               currentState = StatesDocument.Draft;
         }
         break;
      case Published:
         Calendar obj = Calendar.getInstance();
         Date currentDate = obj.getTime();
         System.out.println("Current Date and time: "+currentDate);
         if (data.after(currentDate)) {
            currentState = StatesDocument.Draft;
         }
         break;
   }
}
```

Quins principis SOLID trenca?

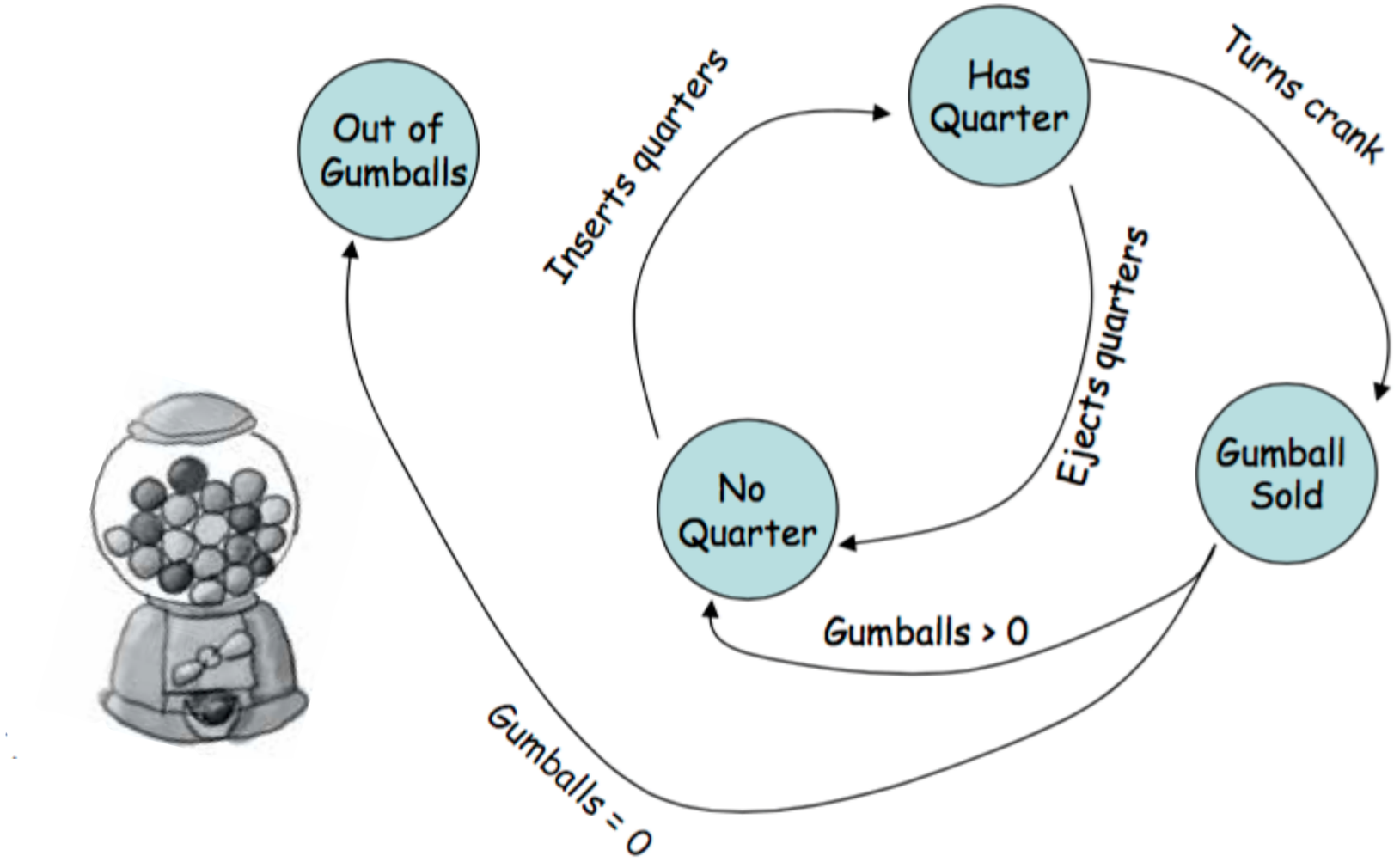
State

- **State** – Patró de disseny de comportament que permet a un objecte alterar el seu comportament quan el seu estat intern canvia. Sembla com si l'objecte canviés la seva classe.

Patró State

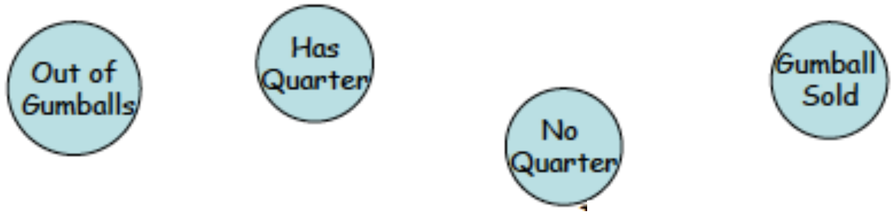

Context té la referència a l'estat actual i delega la funció de canvi d'estat al propi estat. Només coneix la interfície i no estats concrets
1
Context
state
+ Context(initialState)
+ changeState(state)
+ doThis()
+ doThat()
«interface»
State
+ doThis()
+ doThat()
2
Interfície que declara com seran les accions a fer per un estat. En general té el mètde de update() però pot teir d'altres noms o implicar diferents funcions a l'estat que provoquin un canvi d'estat
this.state = state
state.setContext(this)
state.doThis()
ConcreteStates
- context
+ setContext(context)
+ doThis()
+ doThat()
3
Estats concrets que fan el update segons el que ha notificat la classe de Context. Aquests estats concrets tenen un atribut de tipus la classe de Context per que normalment necessiten valors d'aquesta classe Context per determinar el canvi a fer. A més necessiten la classe de Context per a fer el canvi de estat
Client
initialState = new ConcreteState()
context = new Context(initialState)
context.doThis()
// Current state may have been
// changed by context or the state
// object itself.
4
Client és la classe que usarà el context per a poder inicialitzar la màquina d'estats i per a notificar-li els canvis
// A state may issue state
// transition in context.
state = new OtherState()
context.changeState(state)

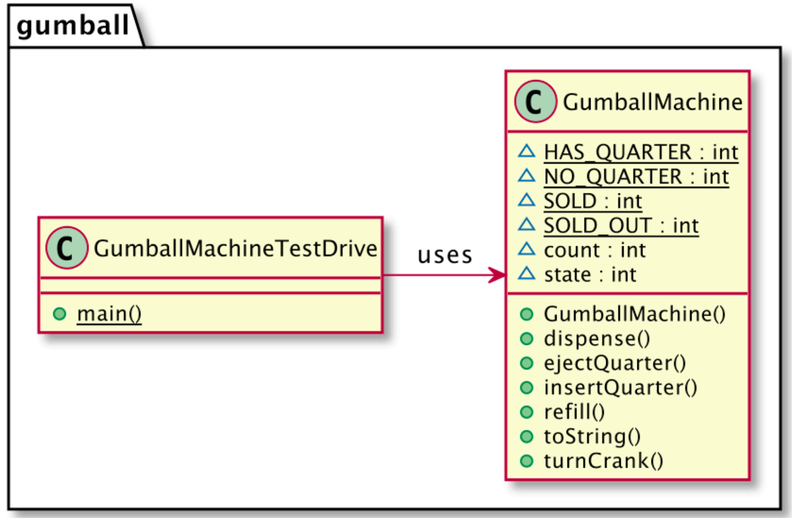
Exemple State : Gumball Machine



Exemple State: Gumball Machine

1. Trobar tots els possibles estats





2. Crear una variable per cada estat i definir un valor per cadascuna d'elles

```
final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;
```

3. Trobar totes les possibles accions que ocorren en el sistema

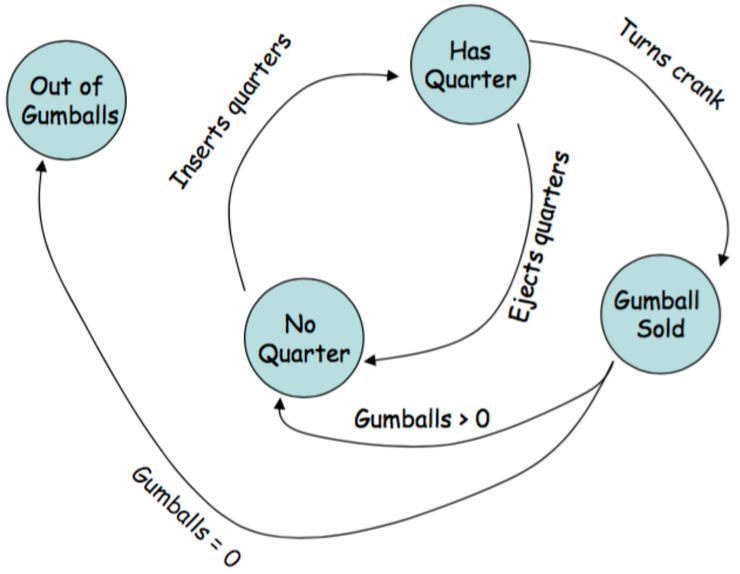
Insert Quarters, **Turns Crank**, **Dispense**, **Eject Quarters**

Example State: Gumball Machine

4. Crear un mètode per a cada acció que actui com una màquina d'estats

```
public class GumballMachine {

      int state = SOLD_OUT;
      int count = 0;

    public void insertQuarter() {
        if (state == HAS_QUARTER) {
            System.out.println("You can't insert another quarter");
        } else if (state == NO_QUARTER) {
            state = HAS_QUARTER;
            System.out.println("You inserted a quarter");
        } else if (state == SOLD_OUT) {
            System.out.println("You can't insert a quarter, the machine is sold out");
        } else if (state == SOLD) {
            System.out.println("Please wait, we're already giving you a gumball");
        }
    }
}
```

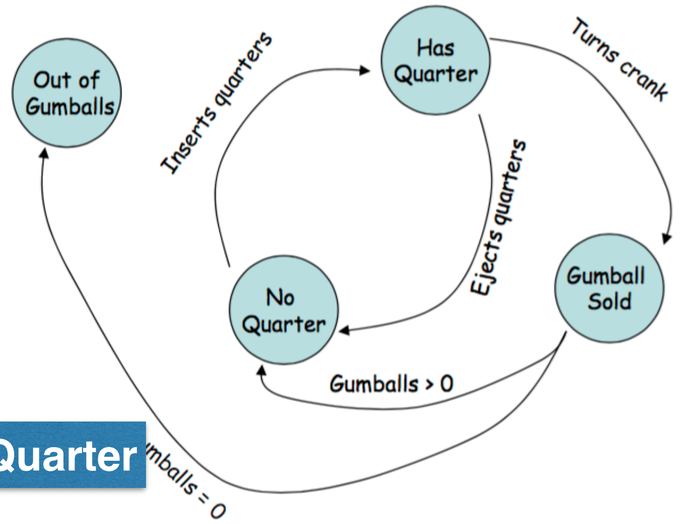
Insert Quarters



Example State: Gumball Machine

```
public void ejectQuarter() {
    if (state == HAS_QUARTER) {
        System.out.println("Quarter returned");
        state = NO_QUARTER;
    } else if (state == NO_QUARTER) {
        System.out.println("You haven't inserted a quarter");
    } else if (state == SOLD) {
        System.out.println("Sorry, you already turned the crank");
    } else if (state == SOLD_OUT) {
        System.out.println("You can't eject, you haven't inserted a quarter yet");
    }
}
```

Eject Quarter

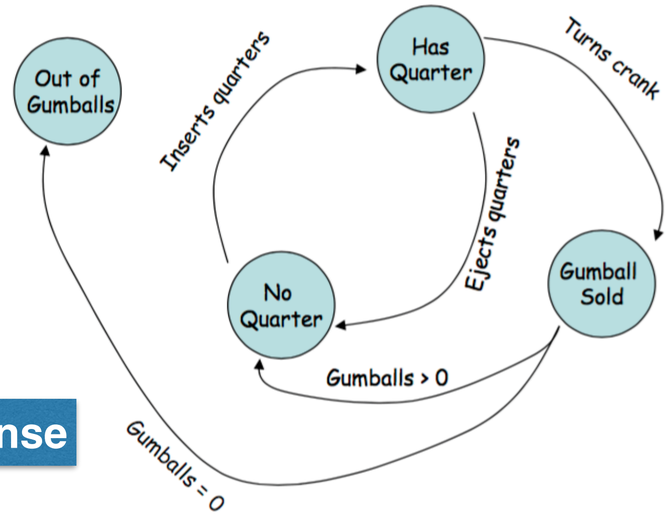



```
public void turnCrank() {
    if (state == SOLD) {
        System.out.println("Turning twice doesn't get you another gumball!");
    } else if (state == NO_QUARTER) {
        System.out.println("You turned but there's no quarter");
    } else if (state == SOLD_OUT) {
        System.out.println("You turned, but there are no gumballs");
    } else if (state == HAS_QUARTER) {
        System.out.println("You turned...");
        state = SOLD;
        dispense();
    }
}
```

Turns Crank

Example State: Gumball Machine

```
public void dispense() {
    if (state == SOLD) {
        System.out.println("A gumball comes rolling out the slot");
        count = count - 1;
        if (count == 0) {
            System.out.println("Oops, out of gumballs!");
            state = SOLD_OUT;
        } else {
            state = NO_QUARTER;
        }
    } else if (state == NO_QUARTER) {
        System.out.println("You need to pay first");
    } else if (state == SOLD_OUT) {
        System.out.println("No gumball dispensed");
    } else if (state == HAS_QUARTER) {
        System.out.println("No gumball dispensed");
    }
}
```



Dispense

Refill

```
public void refill(int numGumBalls) {
    this.count = numGumBalls;
    state = NO_QUARTER;
}
```

quin problema hi ha amb aquesta solució?

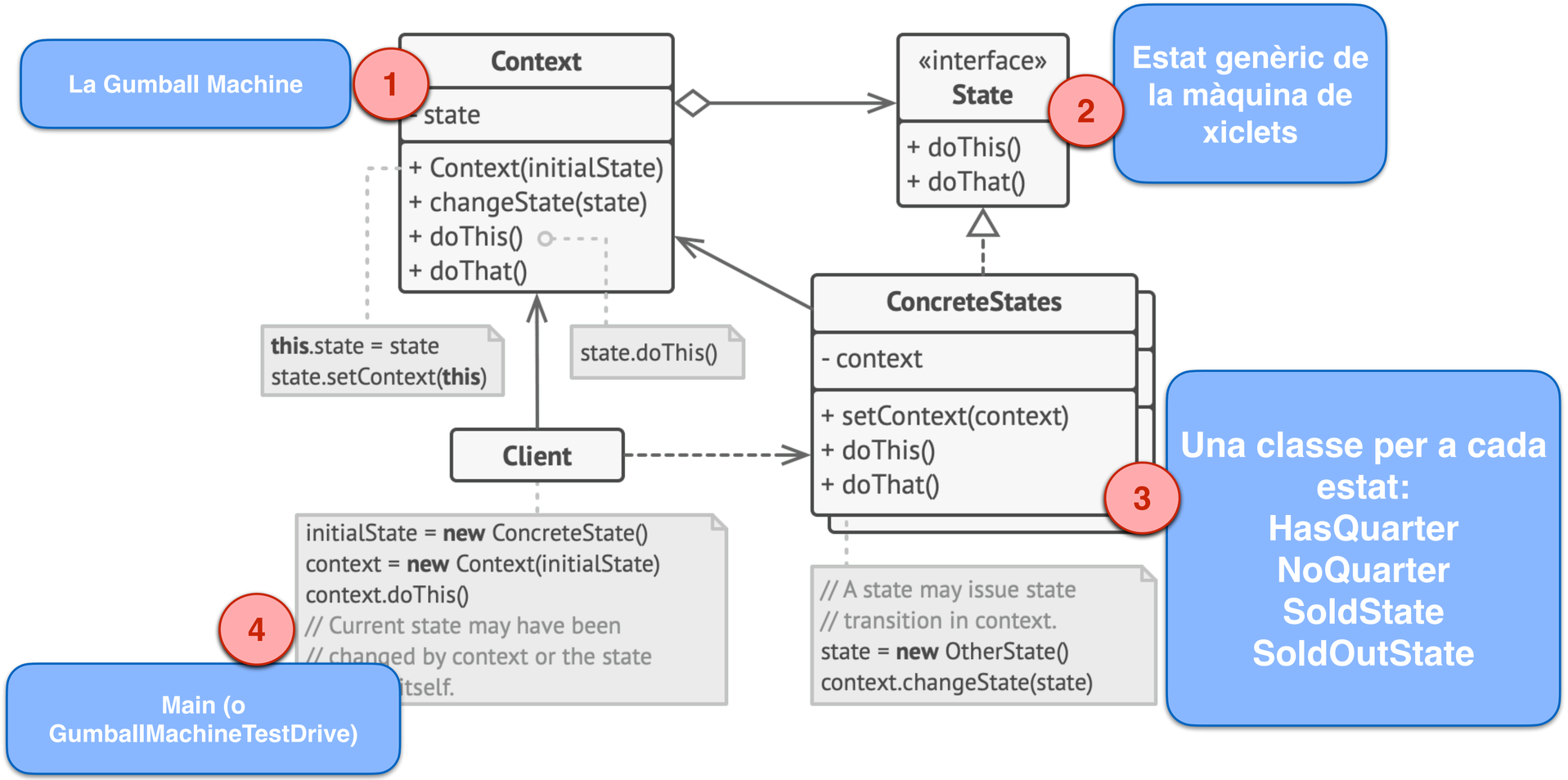
Example State: Gumball Machine

- La solució anterior no permet créixer el nombre d'estats sense modificar el codi ja existent: vulneració OpenClosed

- Solució: Encapsulació dels estats:
  1. Definir la interfície State
  2. Implementar cada classe State per cadascun dels estats de la màquina de xiclets
  3. Posar cada possible acció als estats

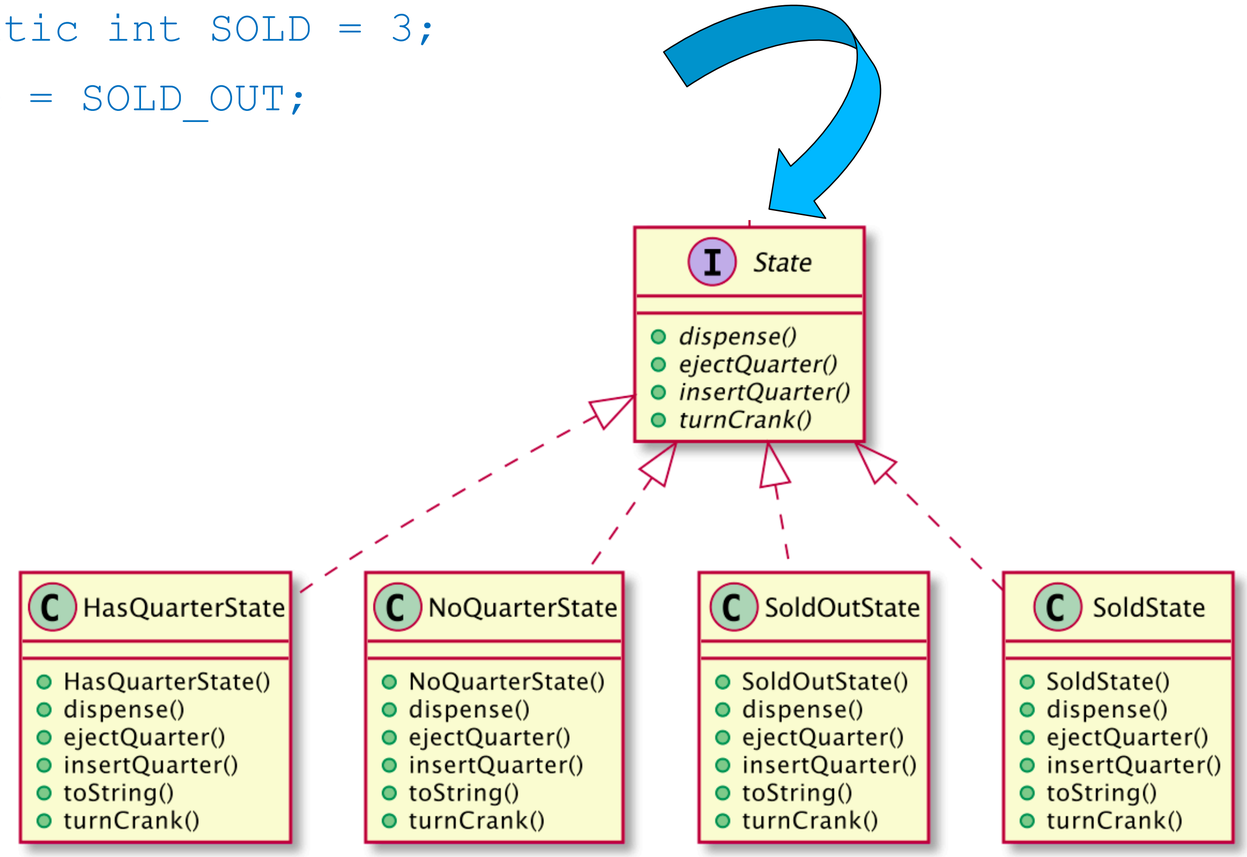
Exercici: Baixa el projecte GumballMachine del Campus i aplica el patró de Màquina d'estats al problema plantejat.

Patró State: GumBall

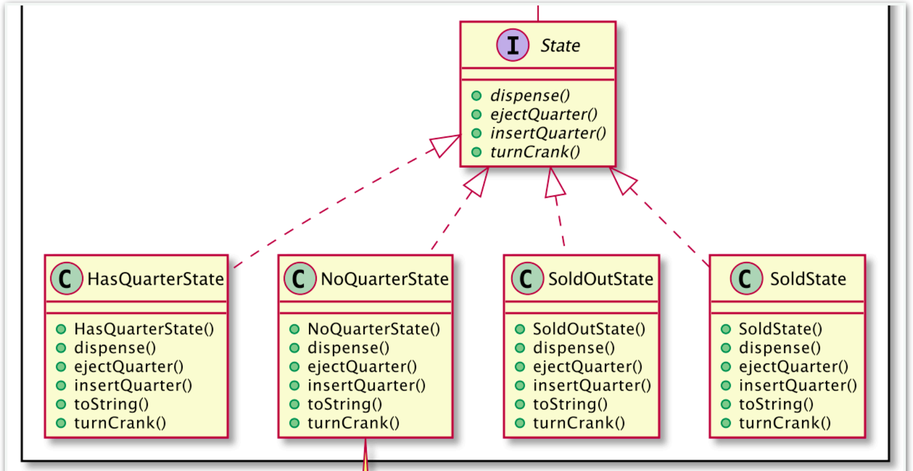


Example State: Gumball Machine

```
final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;

int state = SOLD_OUT;
```



Example State: Gumball Machine




```
public class NoQuarterState implements State {
   GumballMachine gumballMachine;

   public NoQuarterState(GumballMachine gumballMachine) {
      this.gumballMachine = gumballMachine;
   }

      public void insertQuarter() {
            System.out.println("You inserted a quarter");
            gumballMachine.setState(gumballMachine.getHasQuarterState());
      }

      public void ejectQuarter() {
            System.out.println("You haven't inserted a quarter");
      }

      public void turnCrank() {
            System.out.println("You turned, but there's no quarter");
       }

      public void dispense() {
            System.out.println("You need to pay first");
      }

      public String toString() {
            return "waiting for quarter";
      }
}
```

Example State: Gumball Machine

```
public class GumballMachine {

    final static int SOLD_OUT = 0;
    final static int NO_QUARTER = 1;
    final static int HAS_QUARTER = 2;
    final static int SOLD = 3;

    int state = SOLD_OUT;
    int count = 0;
```

s'aplica el patró State implementant una classe diferent per a cada estat i no un enum, com abans

abans d'aplicar el patró

```
public class GumballMachine {

    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;

    State state = soldOutState;
    int count = 0;
```

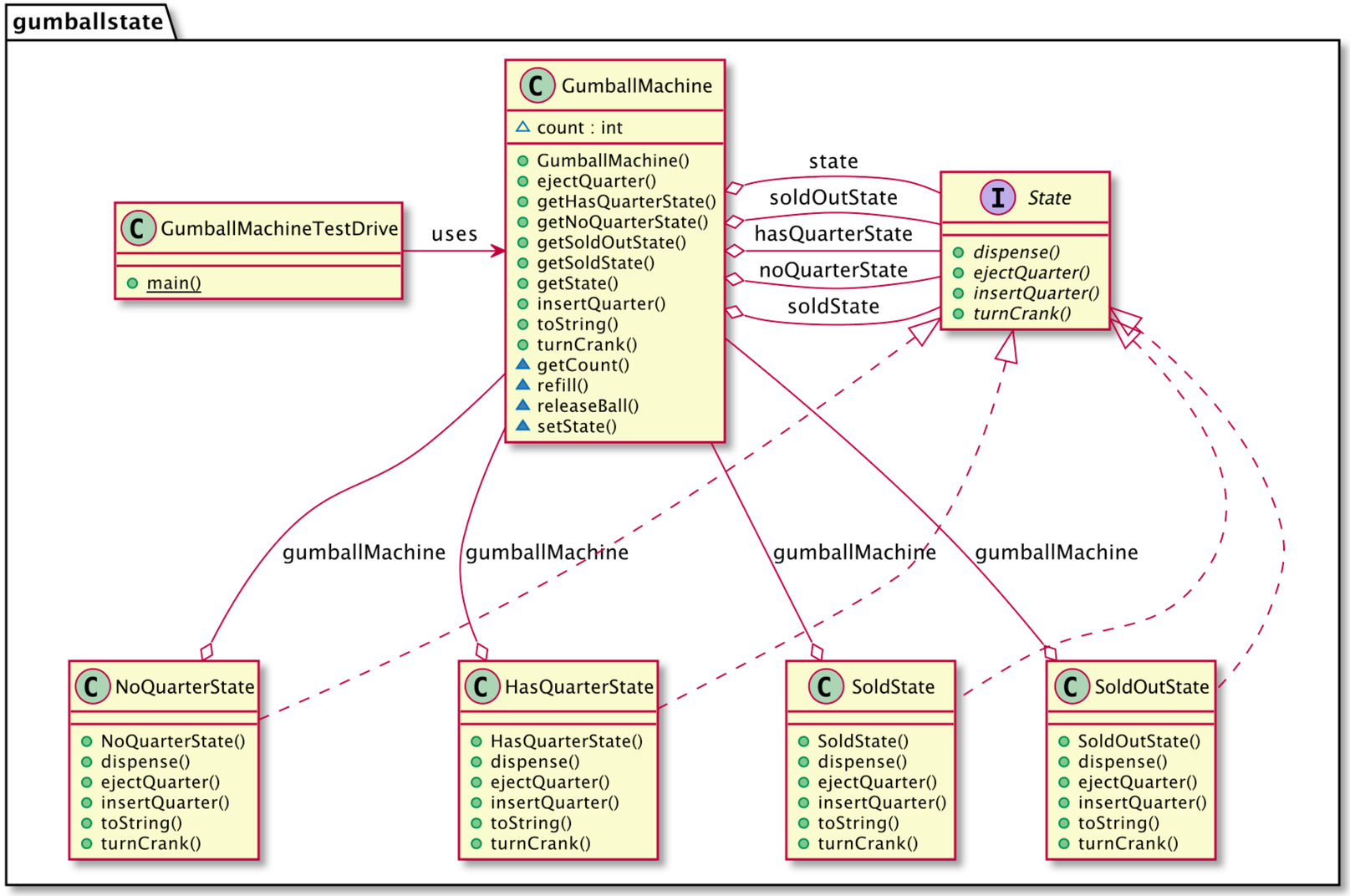
NOU CODI

Tots els estats es creen a l'inici de la màquina de xiclets

l'estat actual és de tipus State i no int com abans

Alternativa als singletons

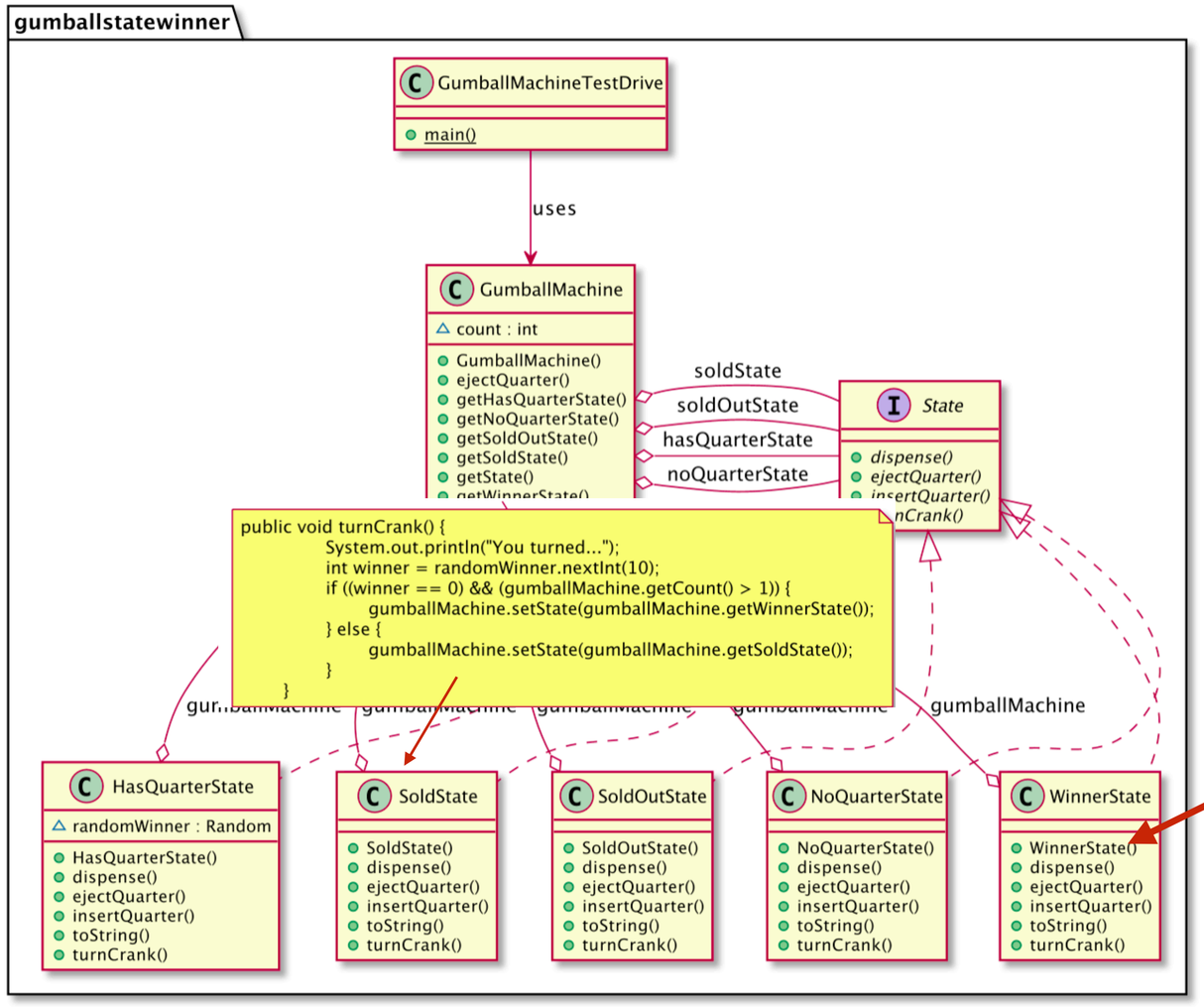
Example State: Gumball Machine



Exemple State: Gumball Machine

Exercici: I si ara es volgués afegir un nou estat en el que es dona premi de forma aleatòria quan es posa una moneda, en el moment de girar la roda per obtenir un xiclet, a l'atzar es poden obtenir 2 xiclets enlloc d'un. Com canviaries el teu projecte?

Example State: Gumball Machine




```
public class WinnerState implements State {
   GumballMachine gumballMachine;

   public WinnerState(GumballMachine gumballMachine) {
      this.gumballMachine = gumballMachine;
   }

      public void insertQuarter() {
            System.out.println("Please wait, we're already giving you a Gumball");
      }

      public void ejectQuarter() {
            System.out.println("Please wait, we're already giving you a Gumball");
      }

      public void turnCrank() {
            System.out.println("Turning again doesn't get you another gumball!");
      }

      public void dispense() {
            System.out.println("YOU'RE A WINNER! You get two gumballs for your quarter");
            gumballMachine.releaseBall();
            if (gumballMachine.getCount() == 0) {
                  gumballMachine.setState(gumballMachine.getSoldOutState());
            } else {
                  gumballMachine.releaseBall();
                  if (gumballMachine.getCount() > 0) {
                        gumballMachine.setState(gumballMachine.getNoQuarterState());
                  } else {
            System.out.println("Oops, out of gumballs!");
                        gumballMachine.setState(gumballMachine.getSoldOutState());
                  }
            }
      }
```