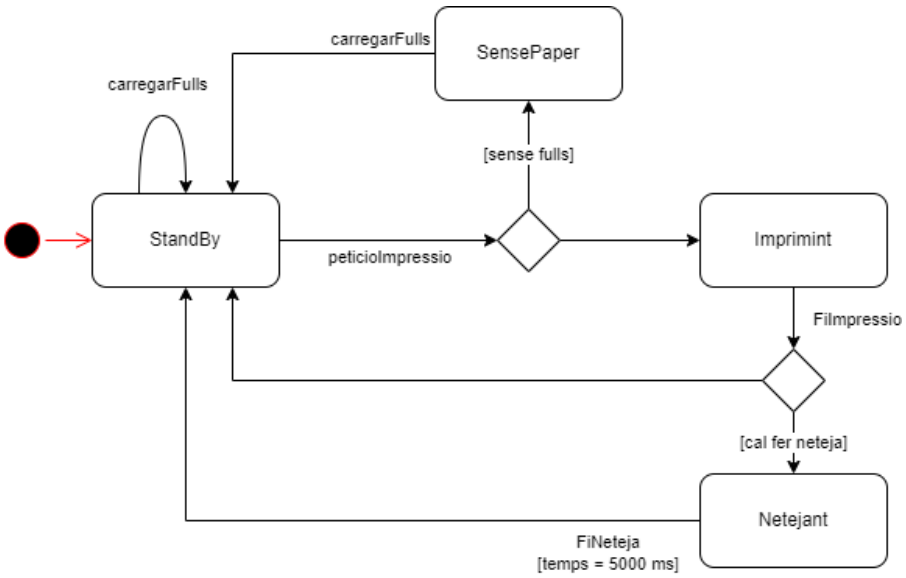
Solució Patró Estats: Impresores

Al TecnoCampus disposem de dues impressores a disposició dels alumnes, volem dissenyar un sistema per poder controlar el seu estat. A continuació, es descriu el funcionament de les impressores:

- Quan l'aparell es connecta es posa en estat "Stan-by" on no fa res.
- Si un alumne demana la impressió d'un document, passa a estat "Imprimint", i en acabar torna a l'estat "Stan-by".
- Però abans de començar una impressió, l'aparell detecta si hi ha prou papers a la safata, si no hi ha s'activa l'estat "SensePaper" i s'envia un missatge a manteniment.
- Al finalitzar una impressió, l'aparell pot detectar que necessita una neteja dels capçals, aquesta neteja és automàtica i dura uns minuts, després torna a l'estat "Stan-by".
- Mentre l'aparell realitza la neteja dels capçals no es pot imprimir.

Diagrama d'Estats

Definim 4 estats: Stand By (que és l'estat inicial), Imprimirnt, Netejant i SensePaper.

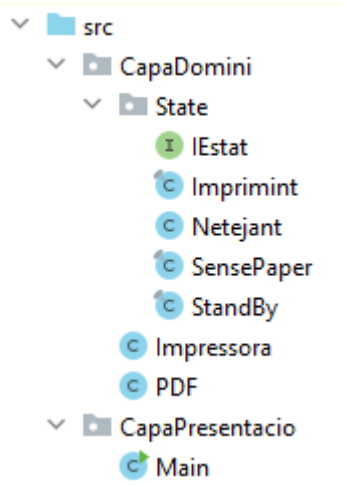


Transiccions

- Petició de impressió (tipus extern)
- Finalització impressió (interna)
- Finalització neteja (temps)
- Carregar fulls (extern)

Patró d'Estats

Es crea el package State dins de la capa domini amb una interfície i 4 estats, apliquem el patró singleton als estats StandBy, SensePaper i Imprimint:




```
package CapaDomini.State;

public interface IEstat {
    public abstract IEstat peticioImpressio(Boolean
potImprimir);
    public abstract IEstat fiImpressio(Boolean
calNeteja);
    public abstract IEstat fiNeteja();
    public abstract IEstat carregarFulls();

    public static IEstat getEstatInicial() {
        return StandBy.getInstanciaUnica();
    }
}
```

```
package CapaDomini.State;

public final class StandBy implements IEstat{
    private static StandBy instanciaUnica = new
StandBy();
    private StandBy(){}

    public static StandBy getInstanciaUnica() {
        return instanciaUnica;
    }

    @Override
    public IEstat peticioImpressio(Boolean potImprimir) {
```

```
        if (potImprimir){
            return Imprimint.getInstanciaUnica();
        }
        return SensePaper.getIntanciaUnica();
    }

    @Override
    public IEstat fiImpressio(Boolean calNeteja) {
        throw new RuntimeException("No es pot fer.");
    }

    @Override
    public IEstat fiNeteja() {
        throw new RuntimeException("No es pot fer");
    }

    @Override
    public IEstat carregarFulls() {
        return this;
    }

    public String toString(){
        return "impressora ready";
    }
}
```

```
package CapaDomini.State;

public class Netejant implements IEstat{

    private final long startTime;

    public Netejant()
    {
        this.startTime = System.currentTimeMillis();
    }

    @Override
    public IEstat peticioImpressio(Boolean potImprimir)
        {
            throw new RuntimeException("Impressora fent
neteja de capçals.");
        }

    @Override
    public IEstat fiImpressio(Boolean calNeteja) {
        throw new RuntimeException("Impressora fent
neteja de capçals.");
    }
```

```
    @Override
    public IEstat fiNeteja() {
        if( System.currentTimeMillis() - this.startTime
>= 5000)
            return StandBy.getInstanciaUnica();
        throw new RuntimeException("Impressora fent
neteja de capçals.");
    }

    @Override
    public IEstat carregarFulls() {
        throw new RuntimeException("Impressora fent
neteja de capçals");
    }

    public String toString(){
        return "impressora en neteja";
    }
}
```

La classe Impressora es modifica per funcionar amb un atribut del tipus de la interfície:

```
package CapaDomini;

import CapaDomini.State.IEstat;

public class Impressora {
    private static final int PAQUET_NOU = 100;
    private int fulls;
    private final String ubicacio;
    private IEstat estatActual;

    public Impressora(String ubicacio) {
        this.ubicacio = ubicacio;
        this.fulls = PAQUET_NOU;
        this.estatActual = IEstat.getEstatInicial();
    }

    public String omplirFulls() {
        try{
            this.estatActual =
this.estatActual.carregarFulls();
            this.fulls += PAQUET_NOU;
            return "S'han afegit fulls";
        }catch (Exception e){
            return "No es pot afegir fulls" +
e.getMessage();
        }
    }
```

```
    public String imprimirDocument(PDF document)
    {
        String missatge = this.toString() + " es vol
imprimir un document de "+document.getFulls()+" pàgines
\n";
        try{
            this.estatActual =
this.estatActual.fiNeteja();
            missatge += "S'ha acabat la neteja \n";
        }catch(Exception e) {
            // no cal fer res
        }

        try{
            this.estatActual =
this.estatActual.peticioImpressio(this.fulls >=
document.getFulls());
            missatge += this.estatActual.toString() + "
\n";
            this.estatActual =
this.estatActual.fiImpressio(Math.random() > 0.8);
            this.fulls -= document.getFulls();
            missatge += "S'ha imprés el document de " +
document.getFulls() + " pàgines\n";
            missatge += this.estatActual.toString() + "
\n";
        }catch(Exception e) {
            missatge += this.estatActual.toString() + "
\n";
        }

        return missatge;
    }

    public String toString()
    { return "Impressora a " + this.ubicacio + " amb " +
this.fulls + " fulls, estat " + this.estatActual; }


}
```

TestCase de JUnit5

```
import CapaDomini.Impressora;
import CapaDomini.PDF;
import CapaDomini.State.IEstat;
import CapaDomini.State.Netejant;
import CapaDomini.State.SensePaper;
import CapaDomini.State.StandBy;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;

import java.lang.reflect.Field;
import static org.junit.jupiter.api.Assertions.*;

@TestMethodOrder(MethodOrderer.Random.class)
public class TestPatroEstat {

    private Impressora unaImpressora;

    @BeforeEach
    public void setUp(){
        unaImpressora = new Impressora("test");
    }

   // Crea dues impressores amb 100 pàgines, comprova que
es poden canviar d'estats i comprova que l'estat de les
dues impressores són independents.
   @Test
   public void testEstatsIndependents(){
       IEstat estatPrimera, estatSegona;

        // agafo una segona impressora i la poso en estat
SenseFulls
       Impressora segonaImpressora = new
Impressora("segona");
       int fulls = (int)
getFieldImpressora(segonaImpressora, "fulls");
       segonaImpressora.imprimirDocument(new
PDF(fulls*2));
       estatSegona = (IEstat)
getFieldImpressora(segonaImpressora, "estatActual");
       assertInstanceOf(SensePaper.class, estatSegona);

       // comprovo que la primera impressora està en
StandBy
       estatPrimera = (IEstat)
getFieldImpressora(unaImpressora, "estatActual");
       assertInstanceOf(StandBy.class, estatPrimera);
```

```
        unaImpressora.imprimirDocument(new PDF(1));

        // a les comparo, no sé l'estat de la primera però
serà diferent a SenseFulls
        estatPrimera = (IEstat)
getFieldImpressora(unaImpressora, "estatActual");
        estatSegona = (IEstat)
getFieldImpressora(segonaImpressora, "estatActual");
        assertNotEquals(estatPrimera, estatSegona);
    }

    //Comprova que l'estat inicial d'una impressora sigui
adequat.
    @Test
    public void testEstatInicial(){
        assertInstanceOf(StandBy.class,
IEstat.getEstatInicial());
        assertInstanceOf(StandBy.class,
getFieldImpressora(unaImpressora, "estatActual"));
    }

    //Comprova que la impressora funciona correctament
quan hi ha suficients pàgines i dona un error quan no hi
ha suficient.
    @Test
    public void testSuficientPagines(){
        int fulls = (int)
getFieldImpressora(unaImpressora, "fulls");
        int PAQUET_NOU = (int)
getFieldImpressora(unaImpressora, "PAQUET_NOU");
        int pages = (int)(Math.random()*PAQUET_NOU) + 1;
        assertEquals(PAQUET_NOU, fulls);
        unaImpressora.imprimirDocument(new PDF(pages));
        fulls = (int) getFieldImpressora(unaImpressora,
"fulls");
        assertEquals(PAQUET_NOU-pages, fulls);
    }

    @Test
    public void testInsuficientPagines(){
        int fulls = (int)
getFieldImpressora(unaImpressora, "fulls");
        unaImpressora.imprimirDocument(new PDF(fulls +
1));
        assertInstanceOf(SensePaper.class,
getFieldImpressora(unaImpressora, "estatActual"));
    }
    //Comprova que la impressora fa neteja de capçals
    @Test
    public void testNeteja(){
        int fulls = (int)
```

```
getFieldImpressora(unaImpressora, "fulls");
        for( int idx = 0; idx < fulls; idx++)
            unaImpressora.imprimirDocument(new PDF(1));
        assertInstanceOf(Netejant.class,
getFieldImpressora(unaImpressora, "estatActual"));
    }
    private Object getFieldImpressora(Impressora
laImpressora, String nomAtribut) {
        try {
            Field lAtribut =
laImpressora.getClass().getDeclaredField(nomAtribut);
            lAtribut.setAccessible(true);
            return lAtribut.get(laImpressora);
        } catch (Exception e) {
            fail(e.getMessage());
        }
        return null;
    }
}
```