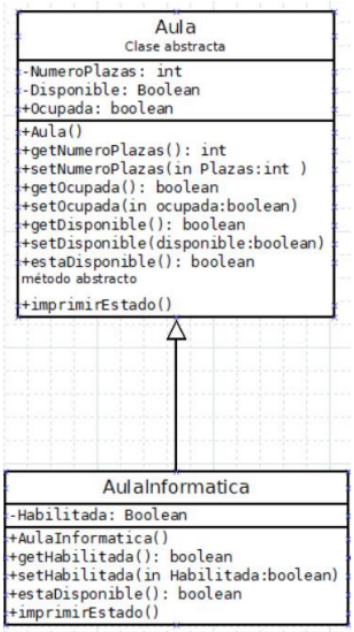
Ejercicio 5 - Overriding

Tenemos la siguiente relación de herencia, representada mediante un diagrama UML:



Hay que realizar las siguientes tareas:

- Implementar las clases:
  - Todos los datos del enunciado se codificarán según lo mostrado en el diagrama UML.
  - El método `imprimirEstado` de la clase `Aula` muestra un mensaje por consola indicando si el aula está disponible o no. La condición de disponible es: `NumeroPlazas > 0` y `Ocupada` es falso.
  - El método `imprimirEstado` de la clase `AulaInformatica` muestra un mensaje por consola indicando si el aula es Apta o no. La condición para que el aula sea Apta es: `NumeroPlazas > 0` y `Ocupada` es falso y `Habilitada` es verdad.
  - Los valores por defecto de los atributos son : `0` y `falso`.
- Implementar el método `estaDiponible()` en la clase `AulaInformatica`.
- Redefinir el método `imprimirEstado()` en la clase `AulaInformatica`.
- Añadir una clase, llamada `Flow`, que tenga el método "`main`". Implementar lo siguiente:
  - Crear un objeto de tipo `AulaInformatica`.
  - Cambiar el número de plazas a `24`.
  - Cambiar el valor del atributo "`Habilitada`" a `true`.
  - Mostrar el estado del `aula de informática`.
  - Cambiar el valor del atributo "`Ocupada`" a `true`.
  - Mostrar otra vez el estado del `aula de informática`.

```
class Flow {

    public static void main(String[] args) {

        AulaInformatica obj = new AulaInformatica();

        obj.setNumeroPlazas(24);
        obj.setHabilitada(true);

        obj.imprimirEstado();

        obj.setOcupada(true);

        obj.imprimirEstado();

    }

}

abstract class Aula {

    private int numeroPlazas;
    private boolean disponible;
    public boolean ocupada;

    public Aula() {

        this.numeroPlazas = 0;
        this.disponible = false;
        this.ocupada = false;

    }

    public int getNumeroPlazas() {
        return this.numeroPlazas;
    }

    public void setNumeroPlazas(int plazas) {
        this.numeroPlazas = plazas;
    }

    public boolean getOcupada() {
        return this.ocupada;
    }

    public void setOcupada(boolean ocupada) {
        this.ocupada = ocupada;
    }

    public boolean getDisponible() {
        return this.disponible;
    }

    public void setDisponible(boolean disponible) {
        this.disponible = disponible;
    }

    public boolean estaDisponible;

    public void imprimirEstado() {
        System.out.println(
                (getNumeroPlazas() > 0 && !getOcupada()) ? "El aula está disponible" : "El aula no está disponible");
    }

}

class AulaInformatica extends Aula {

    private boolean habilitada;

    public AulaInformatica() {
        this.habilitada = false;
    }
```

```
    public boolean getHabilitada() {
        return this.habilitada;
    }

    public void setHabilitada(boolean habilitada) {
        this.habilitada = habilitada;
    }

    public boolean estaDisponible() {
        return super.getDisponible();
    }

    @Override
    public void imprimirEstado() {
        System.out.println(
                (getNumeroPlazas() > 0 && !getOcupada() && getHabilitada()) ? "El aula de informática está disponible"
                        : "El aula de informática no está disponible");
    }

}
```

¿Qué dificultades has encontrado?

Hemos tenido dificultades a la hora de definir la abstracción.

¿Cómo lo has solucionado?

Hemos resuelto el problema después de revisar los apuntes y declarar la clase como abstracta, además de extender la clase `Aula` con `AulaInformatica`.

¿Has obtenido alguna conclusión?

Que el overriding permite a las subclases reemplazar la implementación de métodos de sus superclases. Facilitando la adaptación y personalización del comportamiento de las clases derivadas.