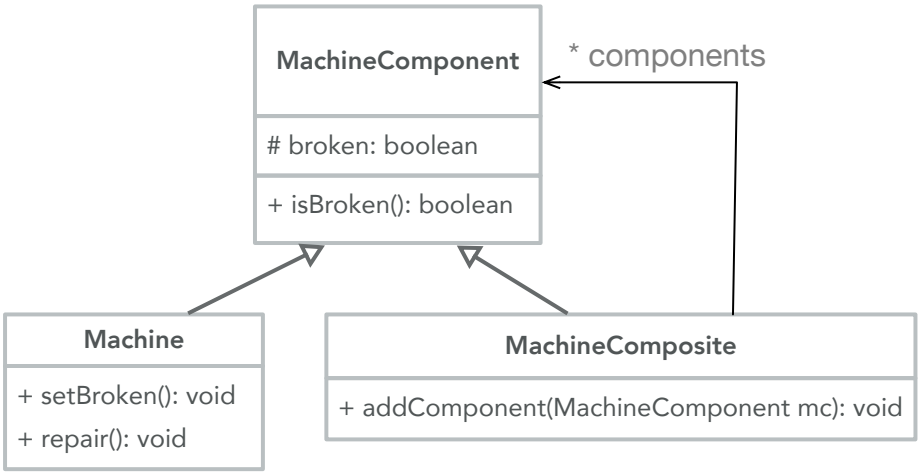
Comenteu mínimament el que preteneu fer i perquè, per a poder valorar millor la vostra solució.

Problema 1 (5 punts)

Donat el següent disseny de partida:



L'estat de trencament d'una máquina és diferent per cada subclasse:

- Una `Machine` està trencada o no trencada per ella mateixa (i, per això, tenim els dos mètodes por canviar el seu estat).
- En canvi, una `MachineComposite` està trencada si algun dels seus components està trencat.

Això seria trivial d'implementar de forma recursiva, però hi ha un problema: cada vegada que es vol saber l'estat de trencament d'un compost caldria travessar l'arbre de subcomponents i això té un cost.

Per això, algú ha pensat en el següent disseny: mantenir a cada component un booleà amb el seu estat, de manera que la implementació del mètode isBroken pot definir-se, **de forma única** a `MachineComponent`, com:

```
1. public class MachineComponent {
2.   protected boolean broken = false;
3.   public final[1] boolean isBroken() { return broken; }
4. }
```

El que es demana és que completeu el disseny per a fer aquest disseny possible i **sense fer un recorregut de subcomponents** per saber si una `MachineComposite` està trencada o no.

En aquest problema, per simplificar, **podeu suposar** que la jerarquia de components **no conté cap cicle**.

[1] Un método final no pot redefinir-se a les subclases i, per tant, la seva implementación ha de ser vàlida per totes elles.

Problema 2 (5 punts)

Partint del disseny inicial del problema 1, afegirem a la classe `Machine` un atribut (`hours`) per saber **les hores de funcionament** i el getter corresponent.

El que voldrem saber és, per cada `MachineComponent`, **és el nombre de `Machine` que han treballat per sobre d´un nombre d'hores donat** (que serà un paràmetre).

El problema és que no ho farem directament, sinó utilitzant el **patró visitor**; per tant caldrà, a més d'implementar el visitor concret que farà aquest càlcul, afegir la infraestructura necessària pel mateix

A diferència del problema anterior, **no podeu suposar** que la jerarquia de components **no conté cap cicle**.