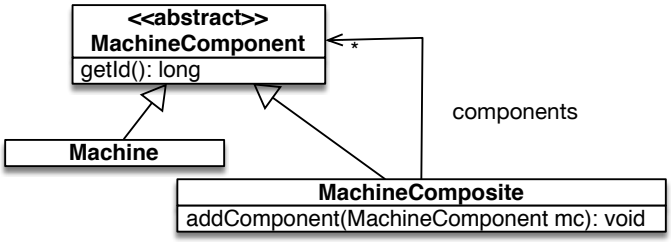
Problema 1. (2 puntos)

- ¿Por qué fue necesario que en Java 8 las interfaces pudiesen tener métodos con implementación?
- ¿Por qué son necesarias las especializaciones primitivas de algunas interfaces funcionales definidas en Java 8?

Problema 2. (5 puntos)

Dado el siguiente diseño:



Se desea poder observar los cambios de composición de las instancias de `MachineComposite`. Es decir, los observadores deberán ser notificados cuando se añade un componente en cualquier nivel de un compuesto. Fijaos en que esto puede pasar:

- directamente, al añadir un subcomponente a un compuesto observado
- indirectamente, al añadir un subsubcomponente a un subcomponente del compuesto observado

Eso sí, los observadores desean saber el componente concreto que directamente ha cambiado su composición (independientemente de si le observan a él o no).

Implementad la clase `MachineComposite` así como un observador que escriba el identificador del componente que ha cambiado su composición:

Tened en cuenta lo siguiente:

- Volem aprofitar la infraestructura de Java. Es decir, la clase `Observable` y la interfaz `Observer`
- No podéis hacer que `MachineComposite` extienda Observable, ya que ja extiende `MachineComponent`
- Tampoco podéis hacer que sea `MachineComponent` quien lo haga, ya que no queremos que las instancia de Machine sean observables.

Por si os es de utilidad, os recuerdo los métodos de la clase Observable (con alguna modificación en la visibilidad de algún método para simplificar el problema):

```
public void addObserver(Observer o) { ... }
public void deleteObserver(Observer o) { ... }
public void deleteObservers() { ... }
public int countObservers() { ... }
public void notifyObservers() { ... }
public void notifyObservers(Object arg) { ... }
public void setChanged() { ... }
public void clearChanged() { ... }
```

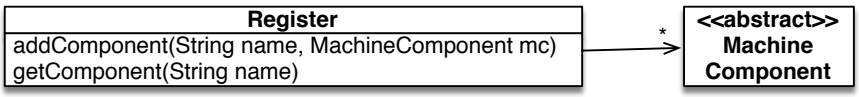
- `public boolean hasChanged() { ... }`

Comentad las decisiones que tomáis; alternativas diferentes que consideráis; … es decir, no escribáis simplemente el código sino explicad brevemente qué habéis considerado para obtenerlo.

Para simplificar, podéis suponer que un componente tiene una estructura arborescente.

Problema 3. (3 puntos)

Queremos poder referirnos a las `MachineComponent` por nombre. Para ello crearemos un registro que asociará a cada nombre, el componente al que se refiere. Es decir:



Pero además querremos que solamente puede hacer una instancia de `Register` en toda la aplicación.

Aplicad el patrón adecuado a la clase `Register`, de **dos formas diferentes (por tanto haciendo dos implementaciones)**, comentando las propiedades que las distinguen.