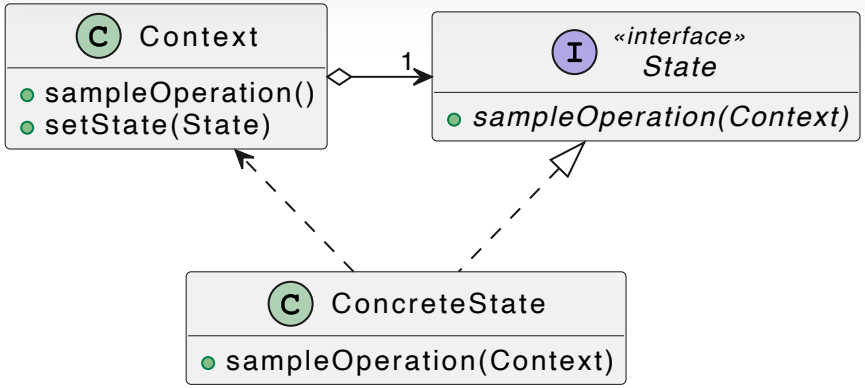
STATE pattern

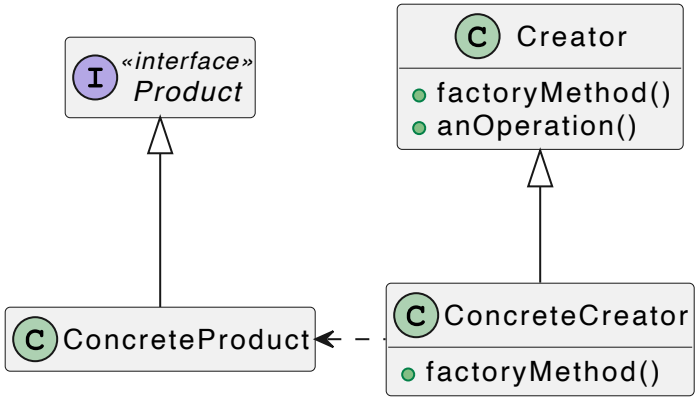
Permette di modellare cambiamenti di comportamento al cambiare dello stato dell'oggetto



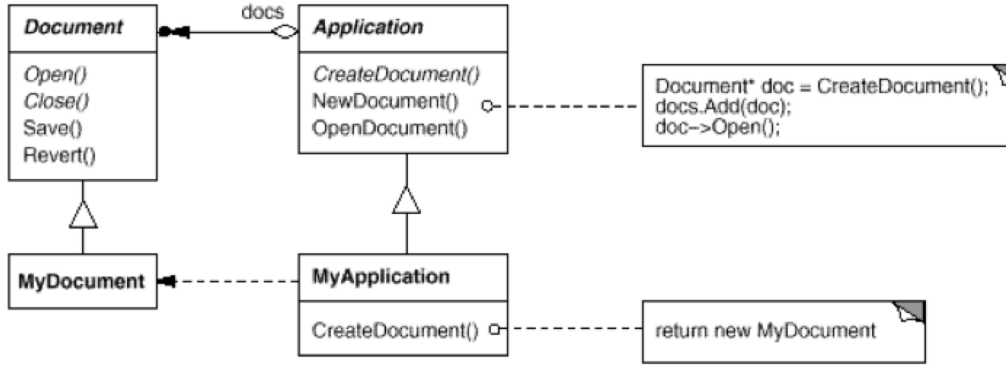

```java
public class Context {
  private State state;
  public void setState(@NotNull State s) {
    state = s;
  }
  public void sampleOperation() {
    state.sampleOperation(this);
  }
}
```

- Viene partizionato il comportamento nelle varie classi stato
- Viene reso esplicito (e atomico) il passaggio di stato
- Gli oggetti stato possono essere in alcuni casi condivisi

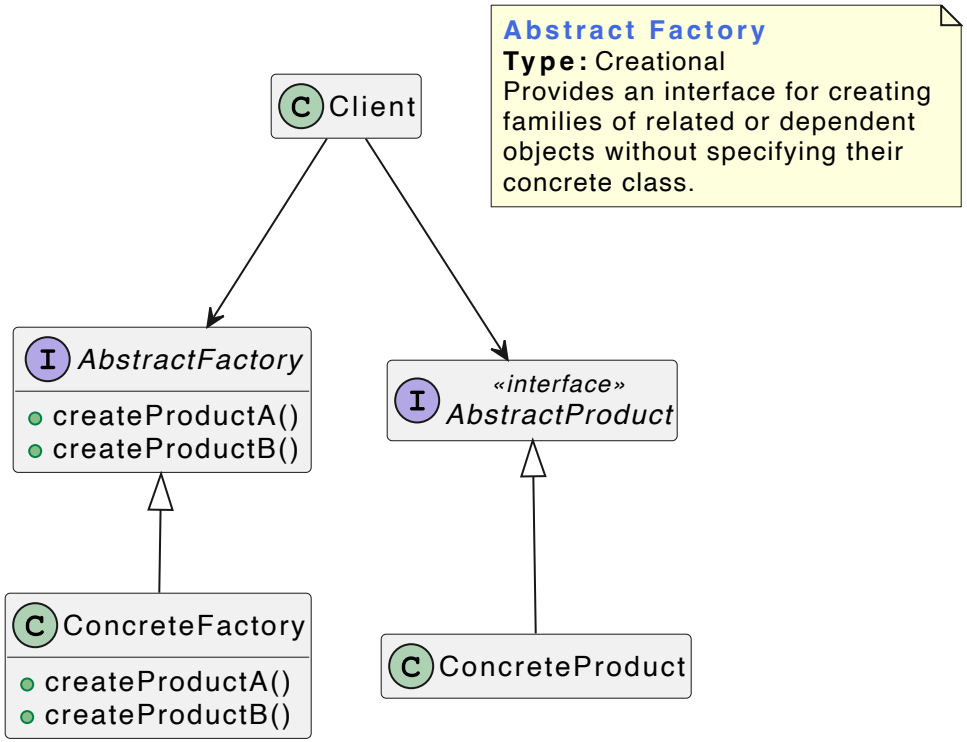
Factory Method pattern



Factory Method
Type: Creational
Definisce una interfaccia per creare un oggetto, ma lascia alle sottoclassi la scelta di cosa creare.



Abstract Factory pattern



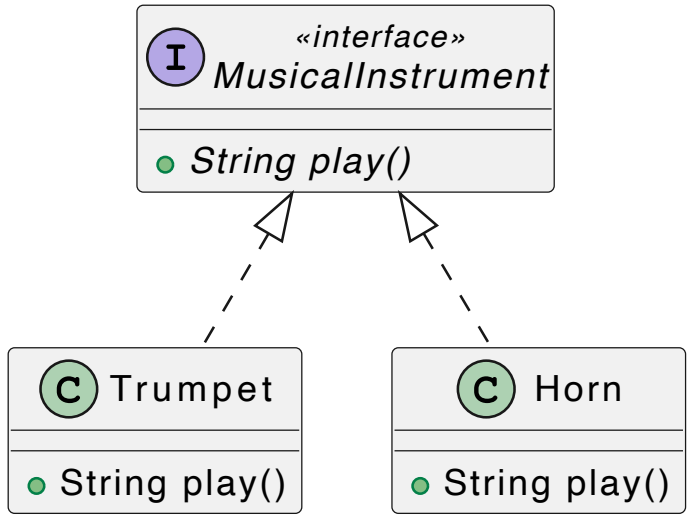
UNIVERSITÀ DEGLI STUDI
DI MILANO

Musica maestro Kata

- Implementare due diverse classi che rappresentano due diversi strumenti: `Trumpet` e `Horn`. Entrambe le classi implementano l'interfaccia `MusicalInstrument` e rispondono alla chiamata del metodo

```
public String play();
```

- che restituisce nel primo caso la stringa **pepepe** e nel secondo la stringa **papapa**.



Implementare altre due classi `WaterGlass` e `IronRod`.

Esse, pur NON implementando l'interfaccia `MusicalInstrument` hanno un comportamento molto simile:

- `WaterGlass` possiede un metodo

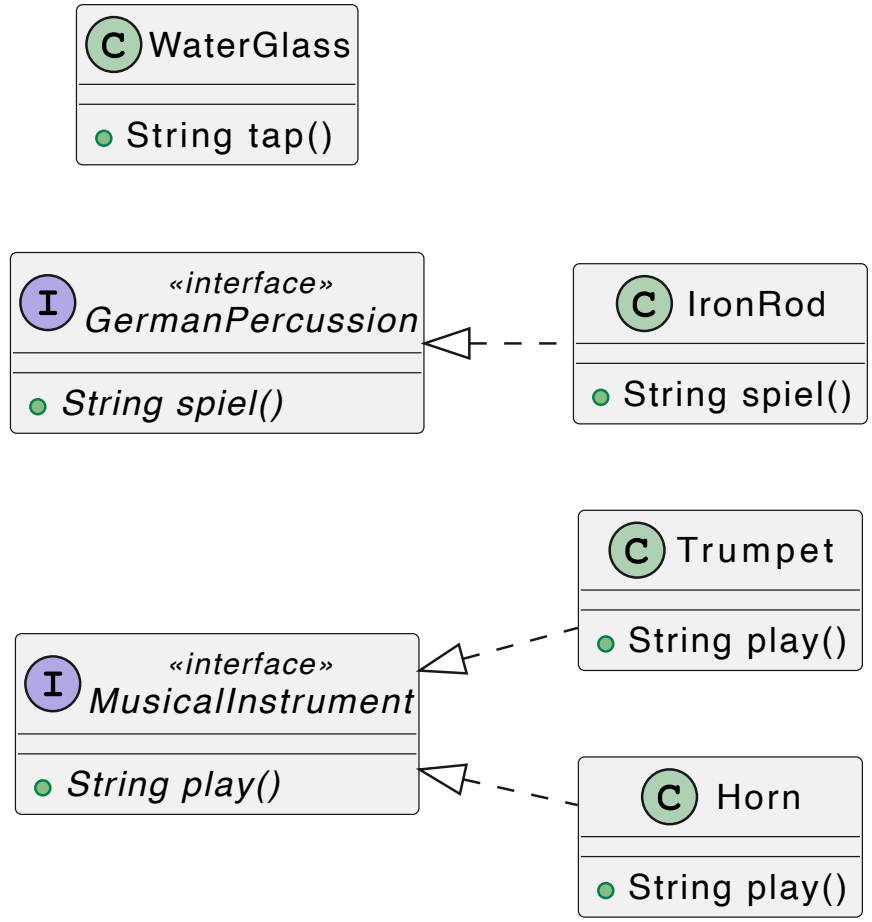
```
public String tap()
```

che restituisce **diding**

- `IronRod` invece aderisce alla interfaccia `GermanPercussion` e risponde quindi alla chiamata

```
public String spiel()
```

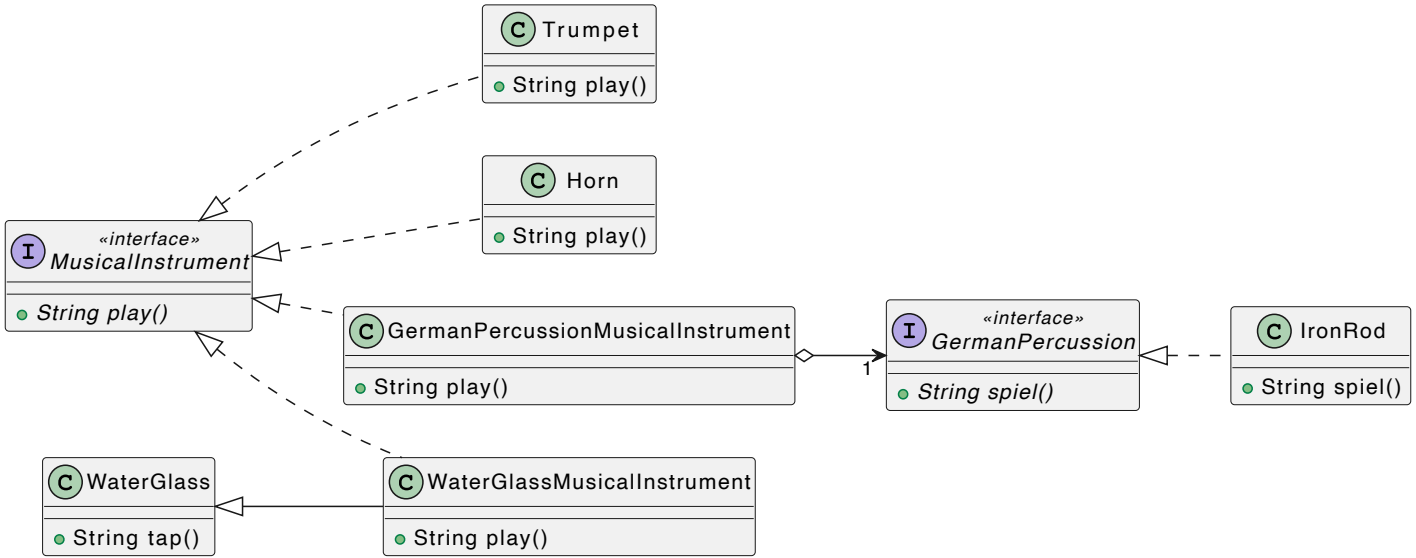
restituendo **tatang**



Aggiungere la possibilità di usare istanze della classe `WaterGlass` e delle classi aderenti alla interfaccia `GermanPercussion` (e in particolare perciò della classe `IronRod`) come oggetti di tipo `MusicalInstrument`

Tale obiettivo deve essere raggiunto usando il design pattern denominato adapter:

- `WaterGlassMusicalInstrument` che realizza un CLASS ADAPTER
- `GermanPercussionMusicalInstrument` che realizza invece un OBJECT ADAPTER.

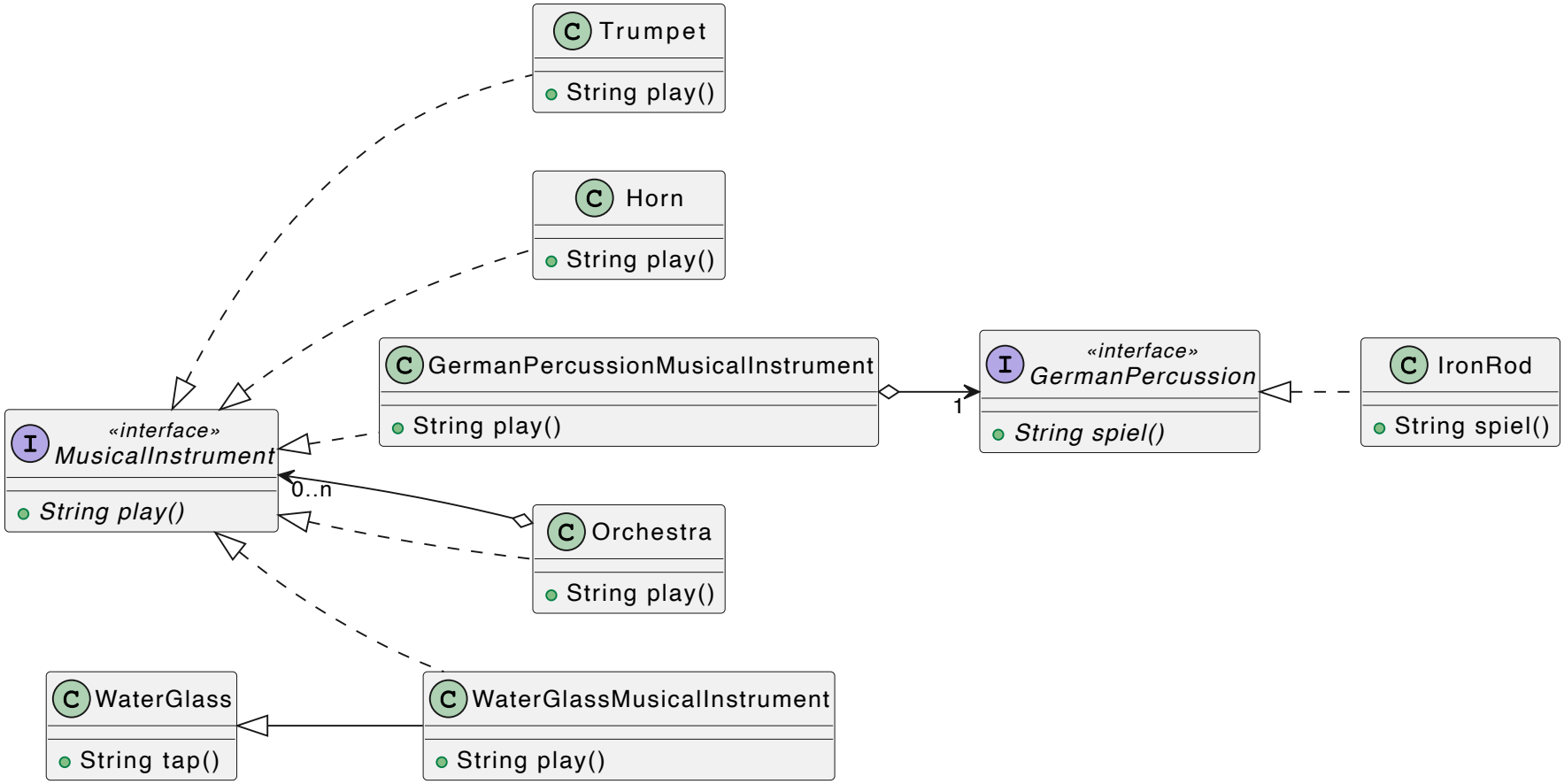


Creare una classe che rappresenti un'orchestra di `MusicalInstrument`.

Tale obiettivo deve essere raggiunto usando il design pattern COMPOSITE

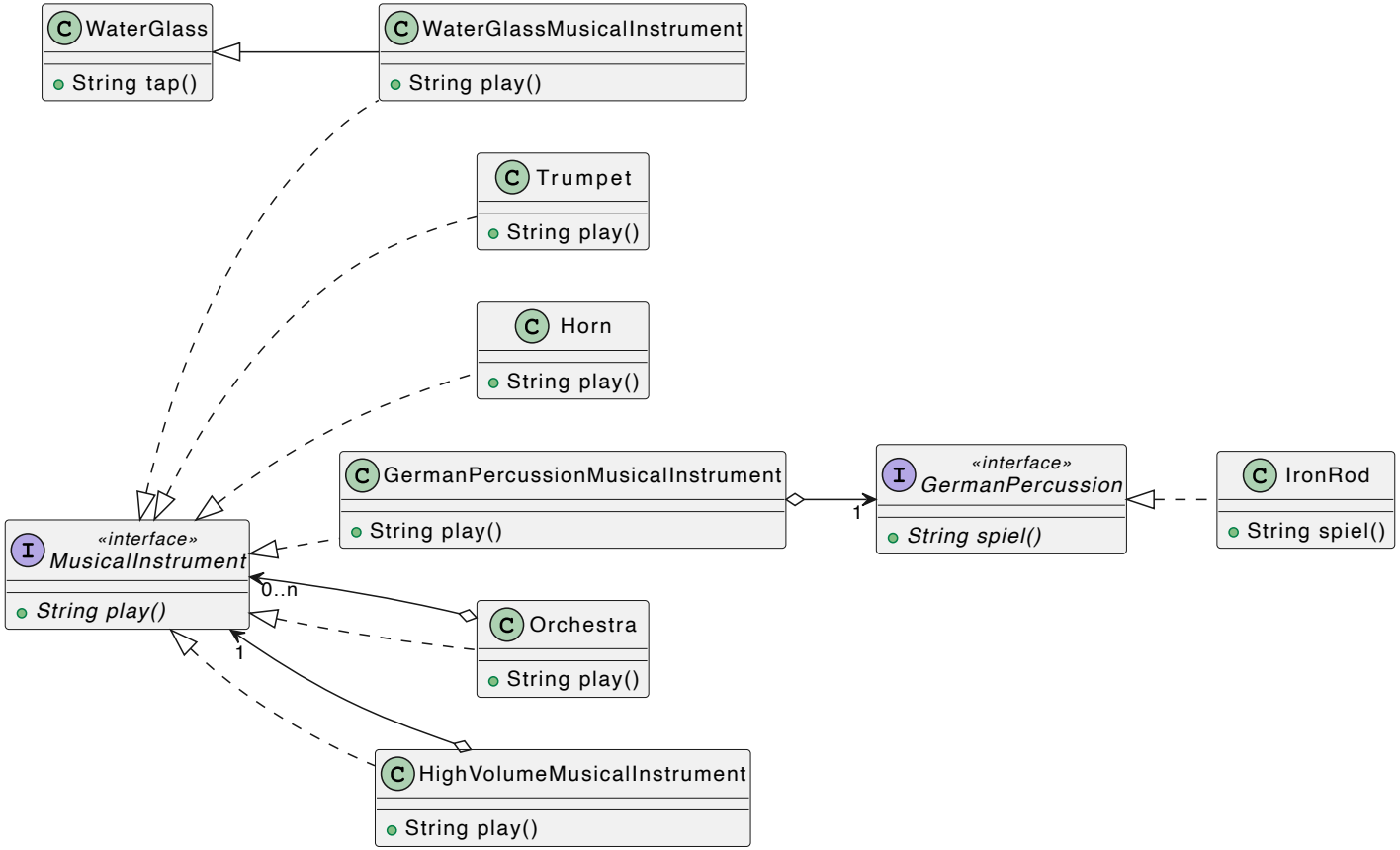
- È richiesta la realizzazione di una classe `Orchestra` che deve implementare l'interfaccia `MusicalInstrument` e deve rispondere all'invocazione del metodo `String play()` demandando la chiamata agli oggetti `MusicalInstrument` aggregati.
- Le istanze della classe `Orchestra` devono consentire l'aggiunta di elementi tramite l'invocazione del metodo

```
public void add(MusicalInstrument instrument)
```

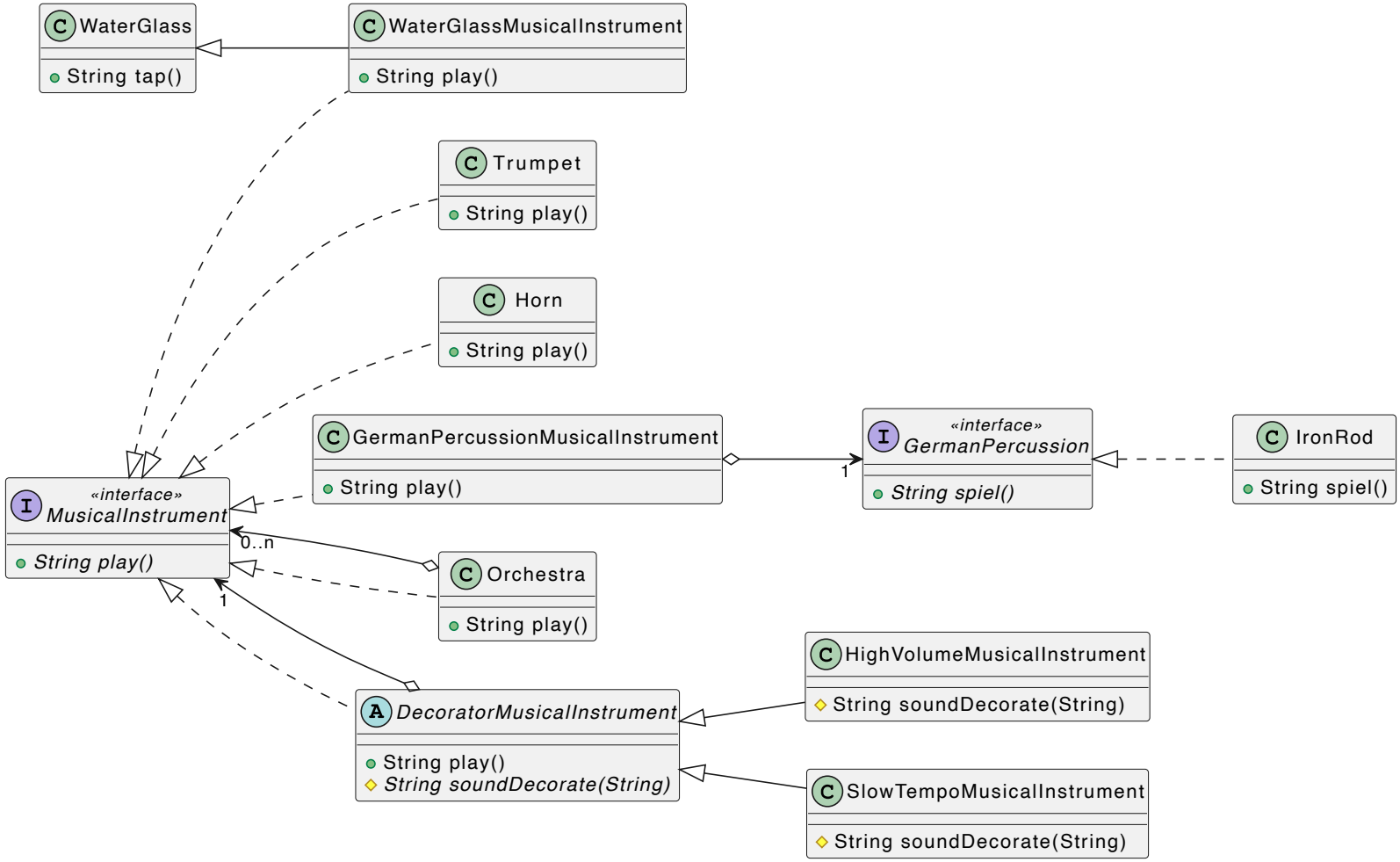

Trumpet
String play()
Horn
String play()
GermanPercussionMusicalInstrument
String play()
«interface»
GermanPercussion
String spiel()
IronRod
String spiel()
1
«interface»
MusicalInstrument
String play()
0..n
Orchestra
String play()
WaterGlass
String tap()
WaterGlassMusicalInstrument
String play()

Aggiungere la possibilità di far suonare più forte (stringa in maiuscolo) un oggetto di tipo `MusicalInstrument`.

Tale obiettivo deve essere raggiunto usando il design pattern DECORATOR; più in dettaglio, è richiesta la realizzazione di una classe `HighVolumeMusicalInstrument` che implementa l'interfaccia `MusicalInstrument` e decora un'istanza di `MusicalInstrument` in modo che tutti gli strumenti decorati quando richiesto suonino "forte".


WaterGlass
String tap()
WaterGlassMusicalInstrument
String play()
Trumpet
String play()
Horn
String play()
GermanPercussionMusicalInstrument
String play()
«interface»
GermanPercussion
String spiel()
IronRod
String spiel()
«interface»
MusicalInstrument
String play()
0..n
1
Orchestra
String play()
1
HighVolumeMusicalInstrument
String play()

- Aggiungere la possibilità di far suonare *più lentamente* (raddoppiando le vocali) un oggetto di tipo `MusicalInstrument`.
- Anche questo obiettivo deve essere raggiunto usando il design pattern DECORATOR creando una classe `SlowTempoMusicalInstrument`
- Dopo averlo realizzato provare a fattorizzare le parti comuni ai due decorator tramite pattern TEMPLATE creando una classe astratta `DecoratorMusicalInstrument`


WaterGlass
String tap()
WaterGlassMusicalInstrument
String play()
Trumpet
String play()
Horn
String play()
GermanPercussionMusicalInstrument
String play()
«interface»
GermanPercussion
String spiel()
1
IronRod
String spiel()
«interface»
MusicalInstrument
String play()
0..n
1
Orchestra
String play()
HighVolumeMusicalInstrument
String soundDecorate(String)
DecoratorMusicalInstrument
String play()
String soundDecorate(String)
SlowTempoMusicalInstrument
String soundDecorate(String)

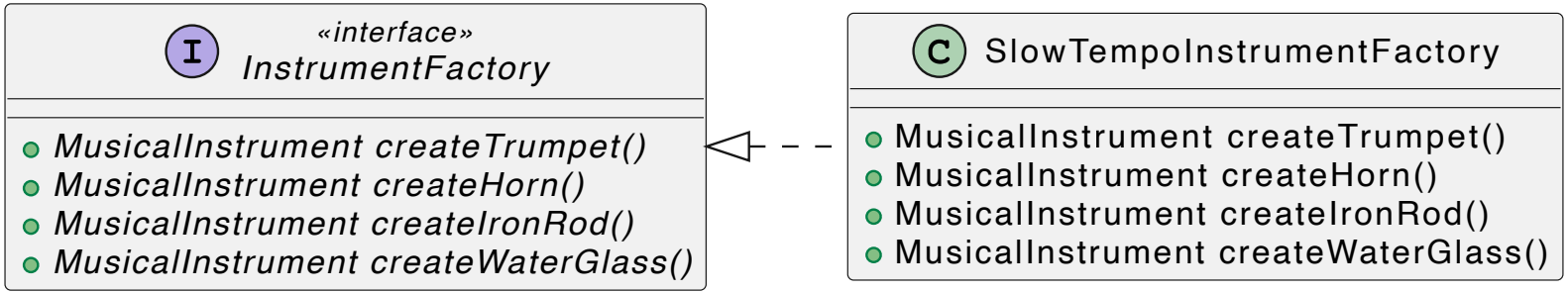
Confusione nella creazione?

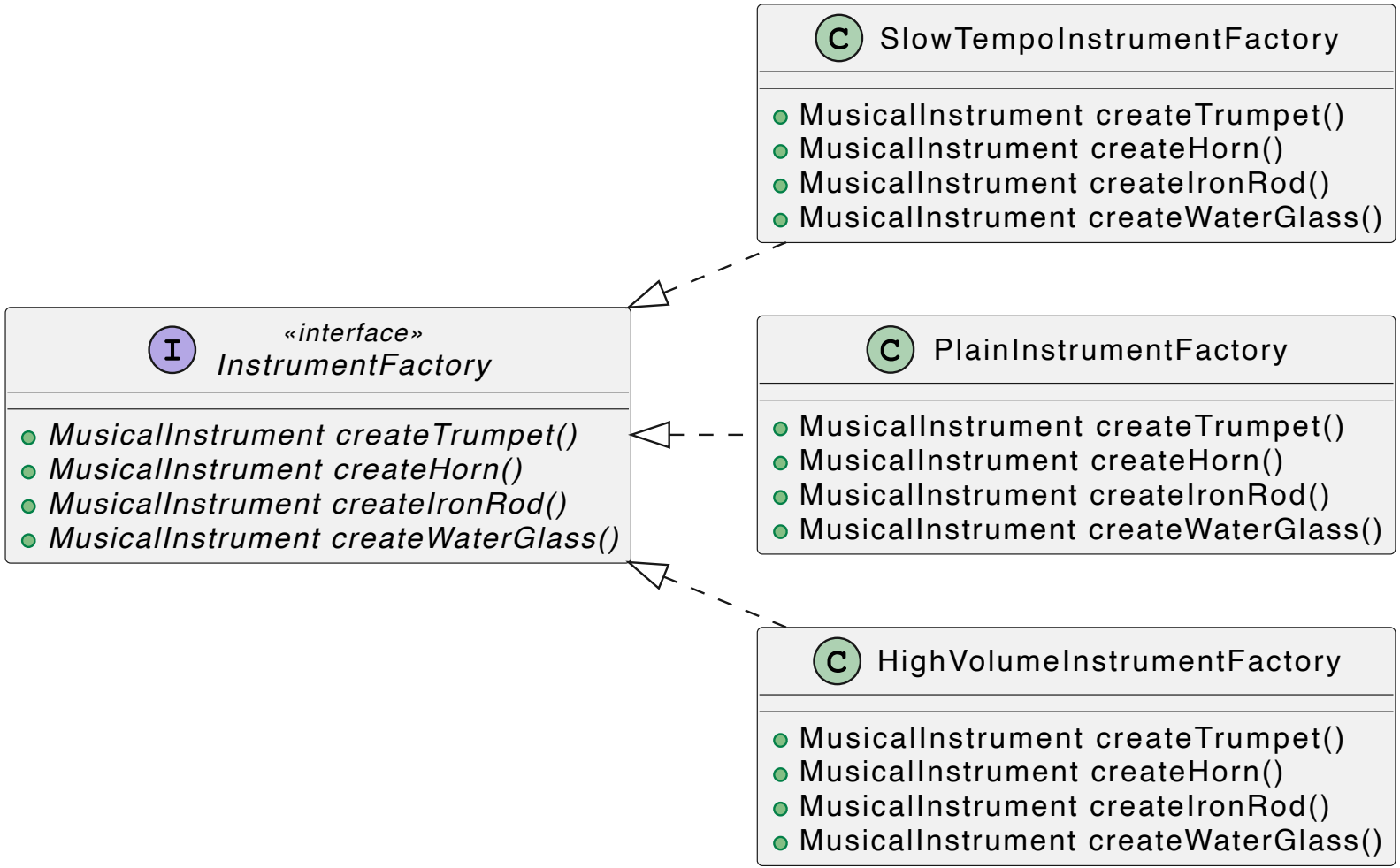
E se mi dimenticassi di dire alla tromba di suonare più lentamente?

Che pattern potremmo usare?

Volendo essere sicuri di creare sempre strumenti che suonano lentamente… vorrei avere un posto unico in cui deciderlo…

Abstract Factory




SlowTempoInstrumentFactory
MusicalInstrument createTrumpet()
MusicalInstrument createHorn()
MusicalInstrument createIronRod()
MusicalInstrument createWaterGlass()
«interface»
InstrumentFactory
MusicalInstrument createTrumpet()
MusicalInstrument createHorn()
MusicalInstrument createIronRod()
MusicalInstrument createWaterGlass()
PlainInstrumentFactory
MusicalInstrument createTrumpet()
MusicalInstrument createHorn()
MusicalInstrument createIronRod()
MusicalInstrument createWaterGlass()
HighVolumeInstrumentFactory
MusicalInstrument createTrumpet()
MusicalInstrument createHorn()
MusicalInstrument createIronRod()
MusicalInstrument createWaterGlass()

Mettiamoci in ascolto

Abbiamo un vicino molto *fastidioso* che ogni volta che sente suonare uno strumento se lo segna e quando interrogato, ci elenca gli strumenti che ha sentito e quante volte.

- vogliamo metterci in ascolto (osservazione): **Observer**
- vogliamo introdurre caratteristica di osservabilità dinamicamente: **Decorator**
- non vogliamo rischiare di dimenticare qualche strumento: **Factory**