

# Design Pattern Facade

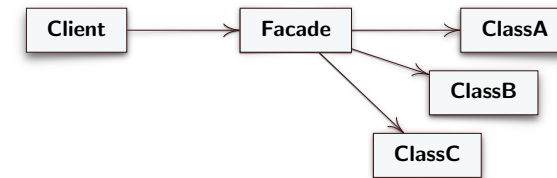
- **Intento:** Fornire un'interfaccia unificata al posto di un insieme di interfacce in un sottosistema (consistente di un insieme di classi). Definire un'interfaccia di alto livello (semplificata) che rende il sottosistema più facile da usare
- **Problema**
  - Spesso si hanno tante classi che svolgono funzioni correlate e l'insieme delle interfacce può essere complesso
  - Può essere difficile capire qual è l'interfaccia essenziale ai client per l'insieme di classi
  - Si vogliono ridurre le comunicazioni e le dipendenze dirette fra i client ed il sottosistema

1

Prof. Tramontana - Aprile 2019

# Design Pattern Facade

- Soluzione
  - **Facade** fornisce un'unica interfaccia semplificata ai client e nasconde gli oggetti del sottosistema, questo riduce la complessità dell'interfaccia e quindi delle chiamate. **Facade** invoca i metodi degli oggetti che nasconde
  - **Client** interagisce solo con l'oggetto Facade



2

Prof. Tramontana - Aprile 2019

# Design Pattern Facade

- Conseguenze
  - Nasconde ai client l'implementazione del sottosistema
  - Promuove l'accoppiamento debole tra sottosistema e client
  - Riduce le dipendenze di compilazione in sistemi grandi. Se si cambia una classe del sottosistema, si può ricompilare la parte di sottosistema fino al facade, quindi non i vari client
  - Non previene l'uso di client più complessi, quando occorre, che accedono ad oggetti del sottosistema
- Implementazione
  - Per rendere gli oggetti del sottosistema non accessibili al client le corrispondenti classi possono essere annidate dentro la classe Facade

3

Prof. Tramontana - Aprile 2019

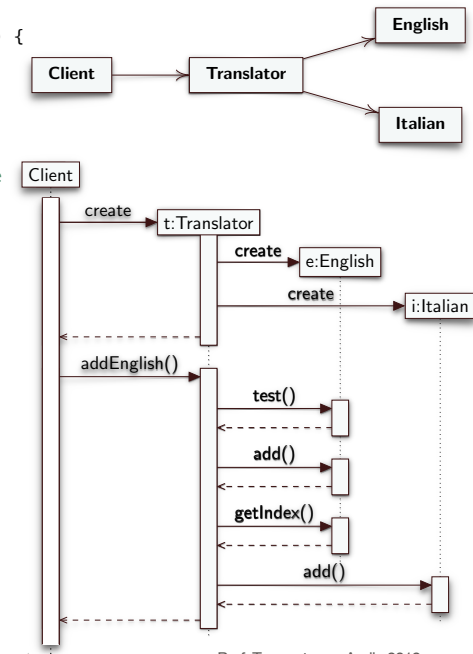
```

public class Client {
    public static void main(String args[]) {
        Translator t = new Translator();
        t.addEnglish("Hello");
        t.multiPrinting();
    }
}

public class Translator { // Ruolo Facade
    private English e = new English();
    private Italian i = new Italian();

    public void addEnglish(String s) {
        if (e.test(s)) {
            e.add(s);
            i.add(e.getIndex(s));
        }
    }

    public void multiPrinting() {
        System.out.print("Italiano: ");
        i.printText();
        System.out.print("English: ");
        e.printText();
    }
}
  
```



4

Prof. Tramontana - Aprile 2019

```

public class English {
    private String text = " ";
    private List<String> d =
        Arrays.asList("Alright", "Hello",
            "Understood", "Yes");

    public boolean test(String s) {
        return d.contains(s);
    }

    public void add(String s) {
        text = text + " " + s;
    }

    public String getText() {
        return text;
    }

    public int getIndex(String s) {
        return d.indexOf(s);
    }

    public void printText() {
        System.out.println(text);
    }
}

```

```

public class Italian {
    private String text = " ";
    private List<String> d =
        Arrays.asList("Va bene", "Ciao",
            "Capito", "Si");

    public void add(int i) {
        text = text + " " + d.get(i);
    }

    public void printText() {
        System.out.println(text);
    }
}

```