

Requisiti

- Il sistema software dovrà fornire la possibilità di prenotare e acquistare un biglietto (per un viaggio). Si potrà annullare la prenotazione, ma non l'acquisto. Ogni biglietto ha un codice, un prezzo, una data di acquisto, il nome dell'intestatario (e i dettagli del viaggio). Per la prenotazione si dovrà dare il nome dell'intestatario.

1

Prof. Tramontana - Aprile 2019

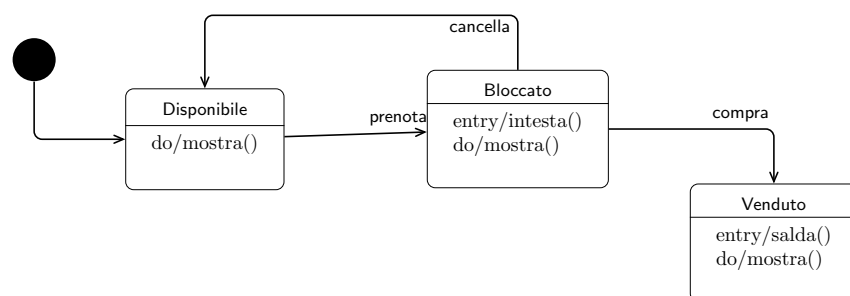
Progettazione

- Il sistema software dovrà fornire la possibilità di **prenotare** e **acquistare** un **biglietto** (per un viaggio). Si potrà **annullare** la prenotazione, ma non l'acquisto. Ogni biglietto ha un **codice**, un **prezzo**, una **data** di acquisto, il **nome** dell'intestatario (e i dettagli del viaggio). Per la prenotazione si dovrà dare il nome dell'intestatario.
- Classi:** Biglietto
- Attributi:** codice, prezzo, data, nome
- Operazioni:** prenota, acquista, annulla
- La classe Biglietto si può trovare in uno degli stati: disponibile, bloccato (ovvero prenotato), venduto

2

Prof. Tramontana - Aprile 2019

Diagramma Degli Stati



- Quindi, la classe Biglietto dovrà implementare i metodi: prenota, cancella, compra, mostra.
- Ciascuna operazione controllerà lo stato in cui si trova il biglietto prima di eseguire le azioni necessarie

3

Prof. Tramontana - Aprile 2019

// Codice che implementa i suddetti requisiti (prima versione)

```
public class Biglietto {
    private String codice = "XYZ", nome;
    private int prezzo = 100;
    private enum StatoBiglietto { DISP, BLOC, VEND }
    private StatoBiglietto stato = StatoBiglietto.DISP;

    // ogni operazione deve controllare in che stato si trova il biglietto
    public void prenota(String s) {
        switch (stato) {
            case DISP:
                System.out.println("Cambia stato da Disponibile a Bloccato");
                nome = s;
                System.out.println("Inserito nuovo intestatario");
                stato = StatoBiglietto.BLOC;
                break;
            case BLOC:
                nome = s;
                System.out.println("Inserito nuovo intestatario");
                break;
            case VEND:
                System.out.println("Non puo' cambiare il nome nello stato Venduto");
                break;
        }
    }
}
```

4

Prof. Tramontana - Aprile 2019

```

public void cancella() {
    switch (stato) {
        case DISP:
            System.out.println("Lo stato era gia' Disponibile");
            break;
        case BLOC:
            System.out.println("Cambia stato da Bloccato a Disponibile");
            nome = "";
            stato = StatoBiglietto.DISP;
            break;
        case VEND:
            System.out.println("Non puo' cambiare stato da Venduto a Disponibile");
            break;
    }
}

public void mostra() {
    System.out.println("Prezzo: " + prezzo + "   codice: " + codice);
    if (stato == StatoBiglietto.BLOC || stato == StatoBiglietto.VEND)
        System.out.println("Nome: " + nome);
}

```

5

Prof. Tramontana - Aprile 2019

```

public void compra() {
    switch (stato) {
        case DISP:
            System.out.println("Non si puo' pagare, bisogna prima intestarlo");
            break;
        case BLOC:
            System.out.println("Cambia stato da Bloccato a Venduto");
            stato = StatoBiglietto.VEND;
            System.out.println("Pagamento effettuato");
            break;
        case VEND:
            System.out.println("Il biglietto era gia' stato venduto");
            break;
    }
}

```

6

Prof. Tramontana - Aprile 2019

```

public class Client {
    private Biglietto b = new Biglietto();
    public static void main(String[] args) {
        usaBiglietto();
    }

    private static void usaBiglietto() {
        b.prenota("Mario Tokoro");
        b.mostra();
        b.compra();
        b.mostra();
    }

    private static void nonUsaOk() {
        b.compra();
        b.cancella();
        b.prenota("Mario Biondi");
    }
}

```

```

Output dell'esecuzione di MainBiglietto
Prezzo: 100   codice: XYZ
Cambia stato da Disponibile a Bloccato
Inserito nuovo intestatario
Prezzo: 100   codice: XYZ
Nome: Mario Tokoro
Cambia stato da Bloccato a Venduto
Pagamento effettuato
Prezzo: 100   codice: XYZ
Nome: Mario Tokoro
Il biglietto era gia' stato venduto
Non puo' cambiare stato da Venduto a Disponibile
Non puo' cambiare il nome nello stato Venduto

```

7

Prof. Tramontana - Aprile 2019

Analisi Del Codice

- La classe ha circa 70LOC, metodo più lungo 15LOC, solo 32 linee con ";"
- Ogni metodo ha vari rami condizionali, uno per ogni stato. La logica condizionale rende il codice difficile da modificare
- Il comportamento in ciascuno stato non è ben separato, poiché lo stesso metodo implementa più comportamenti
- Si può arrivare a un design e un codice più semplice, e che separa i comportamenti? Sì, tramite indiretteeze
- Le condizioni possono essere trasformate in messaggi, questo riduce i duplicati, aggiunge chiarezza e aumenta la flessibilità del codice
- La tecnica di refactoring Replace Conditional with Polymorphism (ovvero Sostituisci i rami condizionali con il polimorfismo), indica come fare
- Ovvero, si tratta del design pattern ... State, ovvero Replace Type Code with State

8

Prof. Tramontana - Aprile 2019

```
// StatoBiglietto e' uno State
public interface StatoBiglietto {
    public void mostra();
    public StatoBiglietto intesta(String s);
    public StatoBiglietto paga();
    public StatoBiglietto cancella();
}

// Disponibile e' un ConcreteState
public class Disponibile implements StatoBiglietto {
    @Override public void mostra() { }

    @Override public StatoBiglietto intesta(String s) {
        System.out.println("DISP Cambia stato da Disponibile a Bloccato");
        return new Bloccato().intesta(s);
    }

    @Override public StatoBiglietto paga() {
        System.out.println("DISP Non si puo' pagare, bisogna prima intestarlo");
        return this;
    }

    @Override public StatoBiglietto cancella() {
        System.out.println("DISP Lo stato era gia' Disponibile");
        return this;
    }
}
```

9

Prof. Tramontana - Aprile 2019

```
import java.time.LocalDateTime;
public class Venduto implements StatoBiglietto { // Venduto e' un ConcreteState
    private final String nome;
    private LocalDateTime dataPagam;

    public Venduto(String n) { nome = n; }

    @Override public void mostra() {
        System.out.println("VEND Nome: " + nome);
    }
    @Override public StatoBiglietto intesta(String s) {
        System.out.println("VEND Non puo' cambiare il nome nello stato Venduto");
        return this;
    }
    @Override public StatoBiglietto paga() {
        if (dataPagam == null) {
            dataPagam = LocalDateTime.now();
            System.out.println("VEND Pagamento effettuato");
        } else {
            System.out.println("VEND Il biglietto era gia' stato pagato");
            return this;
        }
    }
    @Override public StatoBiglietto cancella() {
        System.out.println("VEND Non puo' cambiare stato da Venduto a Disponibile");
        return this;
    }
}
```

11

Prof. Tramontana - Aprile 2019

```
// Bloccato e' un ConcreteState
public class Bloccato implements StatoBiglietto {
    private String nome;

    @Override public void mostra() {
        System.out.println("BLOC Nome: "+nome);
    }

    @Override public StatoBiglietto intesta(String s) {
        System.out.println("BLOC Inserito nuovo intestatario");
        nome = s;
        return this;
    }

    @Override public StatoBiglietto paga() {
        System.out.println("BLOC Cambia stato da Bloccato a Venduto");
        return new Venduto(nome).paga();
    }

    @Override public StatoBiglietto cancella() {
        System.out.println("BLOC Cambia stato da Bloccato a Disponibile");
        return new Disponibile();
    }
}
```

10

Prof. Tramontana - Aprile 2019

```
// Biglietto e' un Context
public class Biglietto {
    private String codice = "XYZ";
    private int prezzo = 100;

    private StatoBiglietto sb = new Disponibile();

    public void mostra() {
        System.out.println("Prezzo: " + prezzo + " codice: " + codice);
        sb.mostra();
    }

    public void prenota(String s) {
        sb = sb.intesta(s);
    }

    public void cancella() {
        sb = sb.cancella();
    }

    public void compra() {
        sb = sb.paga();
    }
}
```

12

Prof. Tramontana - Aprile 2019

```

public class Client {
    private static Biglietto b = new Biglietto();
    public static void main(String[] args) {
        usaBiglietto();
    }
}

```

```

private static void usaBiglietto() {
    b.prenota("Mario Tokoro");
    b.mostra();
    b.compra();
    b.mostra();
}

```

```

private static void nonUsaOk() {
    b.compra();
    b.cancella();
    b.prenota("Mario Biondi");
}
}

```

Output dell'esecuzione di MainBiglietto

```

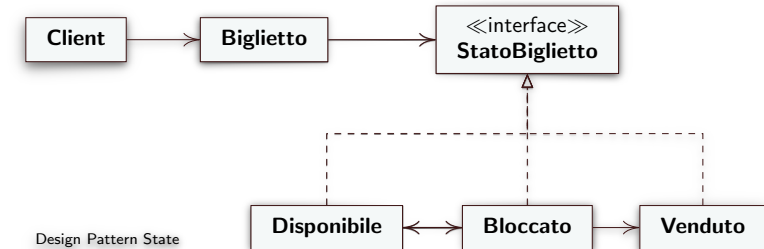
Prezzo: 100 codice: XYZ
DISP Cambia stato da Disponibile a Bloccato
BLOC Inserito nuovo intestatario
Prezzo: 100 codice: XYZ
BLOC Nome: Mario Tokoro
BLOC Cambia stato da Bloccato a Venduto
VEND Pagamento effettuato
Prezzo: 100 codice: XYZ
VEND Nome: Mario Tokoro

VEND Il biglietto era gia' stato venduto
VEND Non puo' cambiare stato da Venduto a Disponibile
VEND Non puo' cambiare il nome nello stato Venduto

```

13

Prof. Tramontana - Aprile 2019



14

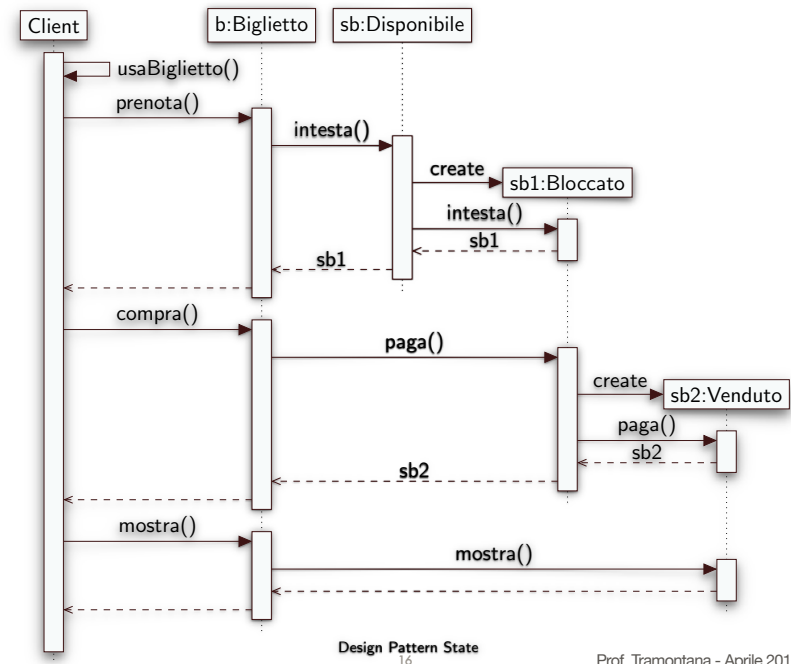
Prof. Tramontana - Aprile 2019

Conseguenze

- Sono state eliminate le istruzioni condizionali: **i metodi non devono controllare in quale stato si trovano**, poiché la classe si riferisce ad un singolo stato. **Non si ha codice duplicato per i test condizionali** su ciascun metodo
- Ogni metodo è più semplice da comprendere e modificare
- **Ciascuno stato avvia, quando occorre, una transizione, questo ha permesso di eliminare l'avvio delle transizioni da Context, e quindi gli switch su esso**
- L'interfaccia usata dai ConcreteState permette di **ritornare il riferimento a un nuovo state** (è detta **fluent**)
- Il codice per ciascuno stato può implementare altre attività senza complicarsi
- **La presenza di switch è un sintomo che suggerisce di usare il polimorfismo**
- Le LOC sono 2 o 3 per metodo, ci sono 4 classi, e 1 interfaccia
- Totale LOC 140 circa (compresi commenti e linee vuote), solo 53 linee con ";"

15

Prof. Tramontana - Aprile 2019



Design Pattern State
16

Prof. Tramontana - Aprile 2019