

# Progettazione del Software

**Simulazione di esame II**

Domenico Fabio Savo

Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti

**Sapienza Università di Roma**

# Requisiti

L'applicazione da progettare riguarda la gestione della vendita di biglietti in un teatro. Ogni spettacolo che si svolge nel teatro è caratterizzato dal titolo (una stringa), dal nome del regista (una stringa) e dal nome degli attori che vi recitano (un insieme non vuoto di stringhe). Per uno spettacolo si prevedono un certo numero di repliche (almeno una), di cui interessa la data e l'ora in cui si svolgono. Dei posti del teatro interessa conoscere la fila, il numero ed il tipo (platea, palco, galleria, ecc.). Alcuni posti sono adatti ad accogliere disabili.

## Requisiti (cont.)

Il costo del biglietto riferito ad un posto (un reale positivo) è stabilito in base al tipo di posto ed allo spettacolo (ad esempio, per lo spettacolo “Cats”, il costo dei posti di platea è di 50 euro). I posti possono essere prenotati. Di una prenotazione  $p$  interessa il nome del cliente (una stringa) che l’ha effettuata, la data in cui è stata effettuata, la replica dello spettacolo a cui  $p$  si riferisce, ed i posti (almeno uno) che sono prenotati tramite  $p$ . Data una prenotazione è di interesse conoscere i posti per disabili eventualmente prenotati.

## Requisiti (cont.)

L'addetto alla biglietteria del teatro è interessato ad effettuare alcuni controlli. In particolare:

- dato uno spettacolo  $s$  ed una tipologia di posto  $t$ , restituire il costo associato alla tipologia  $t$  per lo spettacolo  $s$ , se questo costo è specificato ed è unico. Restituire  $-1$  in caso contrario (questo segnala una situazione di errore);
- dato un insieme  $I$  di posti ed una replica  $r$ , restituire l'insieme dei posti in  $I$  che sono disponibili per  $r$  (cioè i posti contenuti in  $I$  che non risultano prenotati per  $r$ ).

## Requisiti (cont.)

**Domanda 1.** Basandosi sui requisiti riportati sopra, svolgere la fase di analisi producendo lo schema concettuale in UML per l'applicazione e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

**Domanda 2.** Svolgere la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

È obbligatorio solo progettare gli algoritmi e definire le responsabilità sulle associazioni.

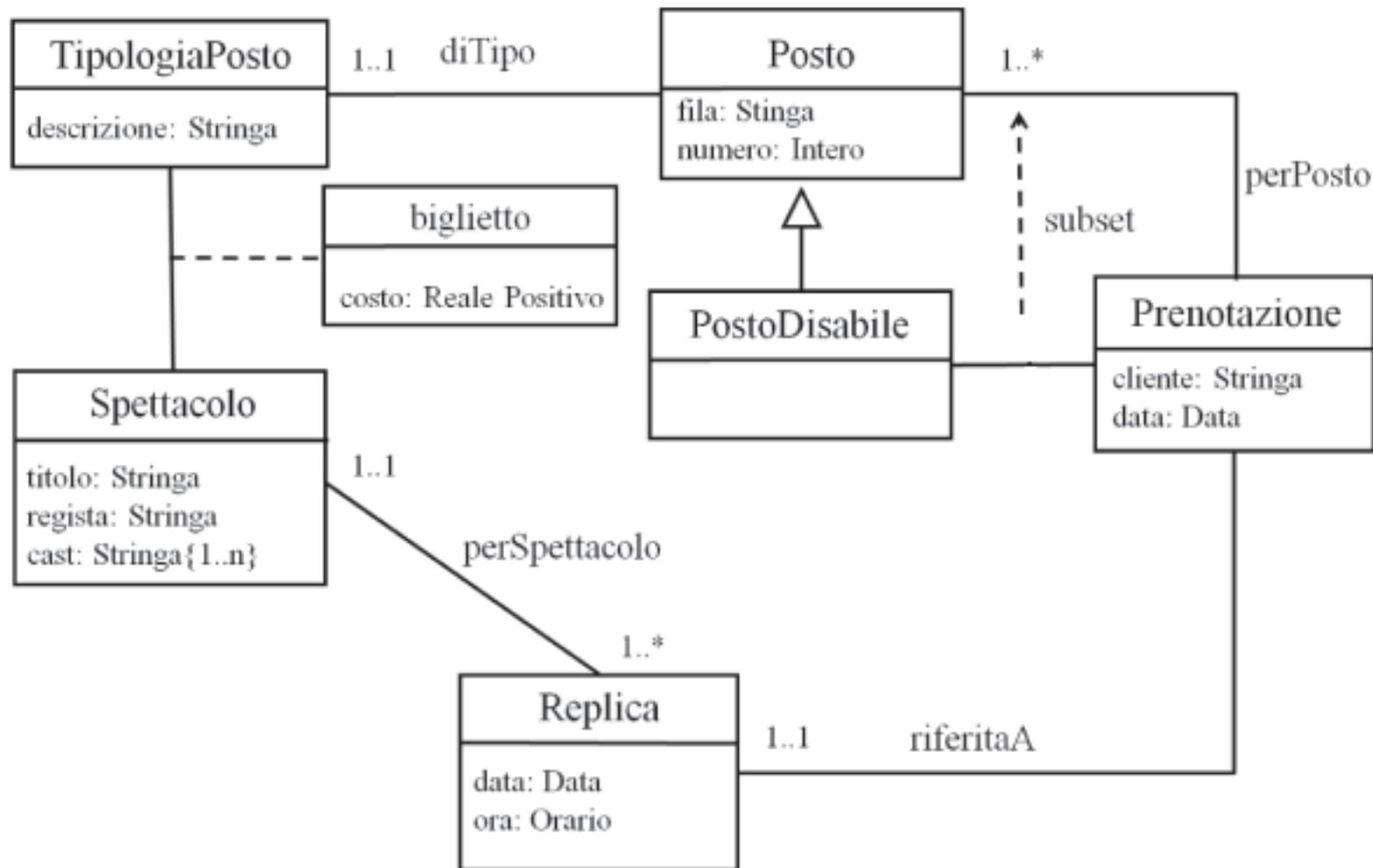
**Domanda 3.** Svolgere la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

È obbligatorio realizzare in Java solo i seguenti aspetti dello schema concettuale:

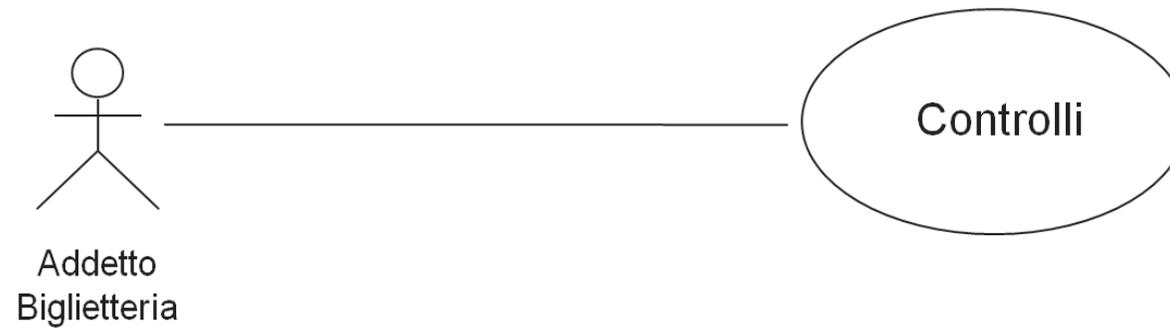
- le classi `TipoPosto` e `Spettacolo` e le associazioni che le legano. Nella realizzazione si ignorino tutte le altre associazioni a cui le classi menzionate partecipano;
- la prima operazione dello use case.

# Fase di analisi

# Diagramma delle classi



# Diagramma degli use case



# Specifica dello use case

## InizioSpecificaUseCase Operazioni

**costoBiglietto** (*s: Spettacolo, t: tipoPosto*): *Reale*

pre: nessuna

post: se non esiste un link di tipo *biglietto* che coinvolge *s* e *t*

result=-1

altrimenti sia  $\ell = \langle s, t \rangle$  un link di tipo *biglietto*

result= *l.costo*

# Specifica dello use case

**postiDisponibili** ( $I : \text{Insieme}\langle \text{Posti} \rangle, r : \text{Replica} \rangle : \text{Insieme}\langle \text{Posti} \rangle$ )

pre: nessuna

post: Si definisca l'insieme  $J$  come segue

$$J = \{p_0 \in I \mid \exists p_r.p_r \in \text{Prenotazione} \wedge \langle p_r, r \rangle \in \text{riferitoA} \\ \wedge \langle p_r, p_0 \rangle \in \text{perPosto}\}$$

result =  $I - J$  (  $-$  denota la differenza insiemistica)

## FineSpecifica

# Fase di progetto

# Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **costoBiglietto**:

```
per ogni link l di tipo biglietto in cui s è coinvolto
  se l.TipoPosto=t
    return l.costo //il link l=<s,t> è unico, quindi appena trovato
                //l'algoritmo termina
return -1; //questo comando viene eseguito solo se non si è trovato
          //il link <s,t> (e cioè, se non è specificato un costo
          //per posti di tipo t per lo spettacolo s)
```

- Per l'operazione **postiDisponibili**:

```
postiPrenotati = new Insieme<Posti>;
per ogni link l di tipo riferitaA in cui r è coinvolto
  per ogni link k di tipo perPosto in cui l.Prenotazione è coinvolta
    se k.Posto appartiene ad I
      aggiungi k.Posto a postiPrenotati;
return I - postiPrenotati;
```

# Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

1. i requisiti,
2. la specifica degli algoritmi per le operazioni di classe e use-case,
3. i vincoli di molteplicità nel diagramma delle classi.

Associazione	Classe	ha resp.
<i>biglietto</i>	<i>TipoPosto</i> <i>Spettacolo</i>	NO SÌ <sup>2</sup>
<i>diTipo</i>	<i>Posto</i> <i>TipoPosto</i>	SÌ <sup>3</sup> NO
<i>perPosto</i>	<i>Prenotazione</i> <i>Posto</i>	SÌ <sup>2,3</sup> NO
<i>perPostoDisabile</i>	<i>Prenotazione</i> <i>PostoDisabile</i>	SÌ <sup>1</sup> NO
<i>riferitaA</i>	<i>Prenotazione</i> <i>Replica</i>	SÌ <sup>3</sup> SÌ <sup>2</sup>
<i>perSpettacolo</i>	<i>Spettacolo</i> <i>Replica</i>	SÌ <sup>3</sup> SÌ <sup>2</sup>

# Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, a causa:

- dei vincoli di molteplicità  $0..*$  delle associazioni,
- delle variabili locali necessarie per vari algoritmi.

Per fare ciò, utilizzeremo l'interfaccia `Set`, e la classe `HashSet` del collection framework di Java 1.5.

# Corrispondenza fra tipi UML e Java

Riassumiamo le nostre scelte nella seguente tabella di corrispondenza dei tipi UML.

Tipo UML	Rappresentazione in Java
Intero	int
Reale Positivo	float*
Stringa	String
Insieme	HashSet
Data	Data
Ora	Ora

\* Con gestione opportuna delle precondizioni.

Si assume di aver definito dei tipi di dato Java Data ed Ora.

# Tabelle di gestione delle proprietà di classi UML

Riassumiamo le nostre scelte differenti da quelle di default mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita*.

<b>Classe UML</b>	<b>Proprietà immutabile</b>
<i>Posto</i>	fila
<i>Posto</i>	numero
<i>Spettacolo</i>	titolo
<i>Spettacolo</i>	regista
<i>Prenotazione</i>	cliente
<i>Prenotazione</i>	data

<b>Classe UML</b>	<b>Proprietà</b>	
	<b>nota alla nascita</b>	<b>non nota alla nascita</b>

# Altre considerazioni

**Sequenza di nascita degli oggetti:** Non dobbiamo assumere una particolare sequenza di nascita degli oggetti.

**Valori alla nascita:** Non sembra ragionevole assumere che per qualche proprietà esistano valori di default validi per tutti gli oggetti.

## Fase di realizzazione

# Considerazioni iniziali

Il compito richiede di realizzare solo quanto segue:

1. la classe UML *TipoPosto*;
2. la classe UML *Spettacolo*;
3. l'associazione UML *biglietto* su cui ha responsabilità la classe *Spettacolo* ma non la classe *TipoPosto*;
4. la prima operazione dello use case.

Procederemo quindi ignorando tutte le altre classi, entità e casi d'uso che non sono fra quelli menzionati sopra.

# Struttura dei file e dei package

```
\---AppTeatro
|  TipoPosto.java
|  Spettacolo.java
|  TipoLinkBiglietto.java
|  Prenotazione.java
|  Replica.java
|  EccezionePrecondizioni.java
|  EccezioneMolteplicita.java
|  Operazioni.java
|
+---Posto
|  Posto.java
|
+---PostoDisabile
|  PostoDisabile.java  |
\---TipiDato
    Data.java
    Ora.java
```

# La classe Java TipoPosto

```
// File AppTeatro/TipoPosto.java
package AppTeatro;

public class TipoPosto {
private String descrizione;

public TipoPosto (String d){
    descrizione = d;
}

public void setTipoPosto(String d){
    descrizione = d;
}

public String getTipoPosto(){
    return descrizione;
}

//due oggetti di classe TipoPosto sono uguali se la loro
//descrizione coincide
public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
        TipoPosto t = (TipoPosto)o;
        return t.descrizione == descrizione;
    }
}
```

```
    else return false;  
  }  
}
```

# La classe Java TipoLinkBiglietto

```
// File AppTeatro/TipoLinkBiglietto.java
package AppTeatro;

public class TipoLinkBiglietto {
    private final Spettacolo loSpettacolo;
    private final TipoPosto ilTipoPosto;
    private final float costo;
    public TipoLinkBiglietto(Spettacolo q, TipoPosto p, float c)
        throws EccezionePrecondizioni {
        if (q == null || p == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        if (c <= 0) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Il costo deve essere un reale positivo");
        loSpettacolo = q; ilTipoPosto = p; costo = c;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkBiglietto b = (TipoLinkBiglietto)o;
            return b.ilTipoPosto == ilTipoPosto &&
                b.loSpettacolo == loSpettacolo;
        }
        else return false;
    }
}
```

```
}  
public int hashCode() {  
    return loSpettacolo.hashCode() + ilTipoPosto.hashCode();  
}  
public Spettacolo getSpettacolo() { return loSpettacolo; }  
public TipoPosto getTipoPosto() { return ilTipoPosto; }  
public float getCosto() { return costo; }  
public String toString() {  
    return "<" + loSpettacolo + ", " + ilTipoPosto + ", " + costo + "Euro >";  
}  
}
```

# La classe Java Spettacolo

```
// File AppTeatro/Spettacolo.java
package AppTeatro;
import java.util.*;

public class Spettacolo {
    private final String titolo;
    private final String regista;
    private HashSet<String> cast;
    private HashSet<TipoLinkBiglietto> biglietti;
    private static final int MIN_ATTORI = 1;

    public Spettacolo(String t, String r) {
        titolo = t;
        regista = r;
        cast = new HashSet<String>();
        biglietti = new HashSet<TipoLinkBiglietto>();
    }

    public String getTitolo() { return titolo; }
    public String getRegista() { return regista; }

    public void InserisciCast(String s) {
        if (s != null)
            cast.add(s);
    }
}
```

```
}

public Set<String> getCast(){
    if(cast.size()>MIN_ATTORI)
        throw new EccezioneMolteplicita("L'insieme di attori che recitano nel film e' vuoto");
    return (HashSet<String>) cast.clone();
}

public void InserisciLinkBiglietto(TipoLinkBiglietto t) {
    if (t != null && t.getSpettacolo()==this)
        biglietti.add(t);
}

public void EliminaLinkBiglietto(TipoLinkBiglietto t) {
    if (t != null && t.getSpettacolo()==this)
        biglietti.remove(t);
}

public Set<TipoLinkBiglietto> getLinkBiglietto(){
    return (HashSet<TipoLinkBiglietto>) biglietti.clone();
}

}
```

# La classe Java Operazioni

```
// File AppTeatro/Operazioni.java
package AppTeatro;
import java.util.*;

public final class Operazioni {
    private Operazioni() { }

    public static float costoBiglietto(Spettacolo s, TipoPosto t) {
        Set<TipoLinkBiglietto> biglietti = s.getLinkBiglietto();
        Iterator<TipoLinkBiglietto> it = biglietti.iterator();
        while(it.hasNext()) {
            TipoLinkBiglietto b = it.next();
            if (b.getTipoPosto() == t)
                return b.getCosto();
        }
        return -1;
    }

    //...
}
```