Sapienza - Università di Roma
Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Esercitazioni di Progettazione del Software - A.A. 2009/2010

Prova al calcolatore del 22 ottobre 2010

Requisiti

Si vuole realizzare un'applicazione che simuli partite del gioco "il tocco del soldato". In ogni partita viene dapprima stabilito un ordine tra i soldati. Quindi, un soldato scelto tra i partecipanti si volta, in modo da non vedere gli altri. A questo punto, il primo tra gli altri soldati decide se toccare il soldato voltato sulla spalla o passare il turno al prossimo soldato il quale, a sua volta, deciderà se toccare o passare il turno (l'ultimo soldato considera come "prossimo" il primo soldato non voltato). La partita procede in questo modo finché qualcuno non tocca il soldato voltato. Quando ciò accade, il soldato voltato accusa chi, secondo lui, lo ha toccato. Se l'accusa è corretta, la partita termina, altrimenti il gioco prosegue.

I *soldati* sono caratterizzati da: *nome*, *cognome* (stringhe) e *numero di matricola* (un intero). Ciascuna *partita* ha una nome ed è giocata da almeno 3 soldati, uno dei quali è *voltato*, che giocano secondo un ordine stabilito in fase di preparazione della partita. Un soldato può giocare in al più una partita per volta. In Figura 1(a), è mostrato il diagramma delle classi corrispondente al dominio.

Un soldato è inizialmente nello stato *innocente*. Da esso, può ricevere diversi eventi:

- evento *turno*: è il turno del soldato corrente, che decide (casualmente) se toccare il soldato voltato o meno. Se lo tocca, gli invia l'evento *tocco* ed entra nello stato *colpevole*; altrimenti, invia l'evento *turno* al prossimo soldato non voltato (secondo l'ordine stabilito nella partita), rimanendo nello stato *innocente*;
- evento *prosegui* (ciò accade quando un innocente è stato accusato): il soldato rimane nello stato *innocente*;
- evento *accusa*: il soldato (accusato ingiustamente) rimane nello stato *innocente* ed invia l'evento *prosegui* a tutti gli altri;
- evento *tocco*: il soldato è stato toccato (solo un soldato voltato può essere toccato, v. sopra). Procede quindi ad accusare chi, secondo lui, è il colpevole (scelto casualmente), inviandogli l'evento *accusa* e rimanendo nello stato *innocente*;
- evento *fine*: la partita è finita, quindi il soldato entra nello stato finale *finita*.

Quando è nello stato *colpevole*, il soldato può ricevere i seguenti eventi:

- evento *prosegui* (ciò accade quando un innocente è stato accusato): il soldato torna nello stato *innocente* ed invia l'evento *turno* al suo successore (non voltato).
- evento *accusa*: il colpevole è stato scoperto, quindi la partita termina. Il soldato, pertanto, invia l'evento *fine* a tutti gli altri, ed entra nello stato finale *scoperto*.

In Figura 1(b), è mostrato il diagramma degli stati e delle transizioni della classe *Soldato*.

L'attività di creazione ed esecuzione della partita è mostrata in Figura 1(c). Innanzitutto, viene chiesto all'utente di inserire il nome della nuova partita, che viene quindi creata. Dopodiché l'attività entra nel ciclo per l'inserimento dei partecipanti, dal quale l'utente potrà scegliere di uscire solo dopo aver inserito almeno 3 soldati ed aver indicato un soldato voltato. Ciascuna iterazione del ciclo è così composta:

- viene chiesto all'utente, tramite opportuna attività di I/O, di inserire i dati relativi al prossimo soldato da inserire, quindi viene creato il nuovo soldato;
- se non è stato ancora specificato il soldato voltato, si chiede all'utente se il soldato appena creato è voltato, altrimenti viene considerato non voltato;
- il nuovo soldato viene inserito nella partita, eventualmente voltato, coerentemente con il passo precedente;
- se alla partita corrente partecipano almeno 3 soldati ed è stato specificato chi tra essi sia voltato, allora viene chiesto all'utente se abbia intenzione di inserire un nuovo soldato;
- se alla partita corrente partecipano meno di 3 soldati o non è ancora stato specificato un soldato voltato, oppure l'utente ha intenzione di inserire un altro soldato, allora viene eseguita una nuova iterazione del ciclo, altrimenti si esce.

L'ordine d'inserimento dei soldati corrisponde all'ordine di gioco. Dopo il completamento del ciclo viene avviata la partita e, al suo termine, vengono stampati su schermo i dati del soldato riconosciuto colpevole.

La prova consiste nel completare o modificare il codice fornito insieme al testo, in modo da soddisfare i requisiti sopra riportati. Seguendo le indicazioni riportate nei commenti al codice, si chiede di intervenire sulle seguenti classi:

- `Main` (package `applicazione`)
- `Ascoltatore` (package `applicazione`)
- `Partita` (package `partita`)
- `ManagerPartecipa` (package `partecipa`)
- `SoldatoFired` (package `soldato`) – il comportamento delle funzioni ausiliarie è opportunamente descritto nei commenti al codice della classe
- `AttivitaPrincipale` (package `attivita_composte`)
- `AttivitaAggiungiAPartita` (package `attivita_atomiche`)

Tempo a disposizione: **3 ore.**

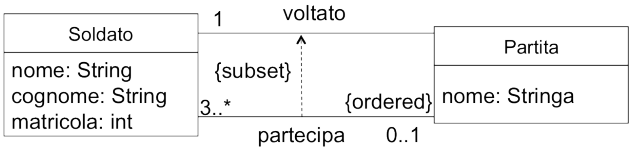
Gli elaborati non accettati dal compilatore saranno considerati insufficienti.

Per facilitare la comprensione del codice e lo svolgimento della prova, nel seguito sono riportati i documenti di specifica risultanti dalle fasi di analisi e di progetto.

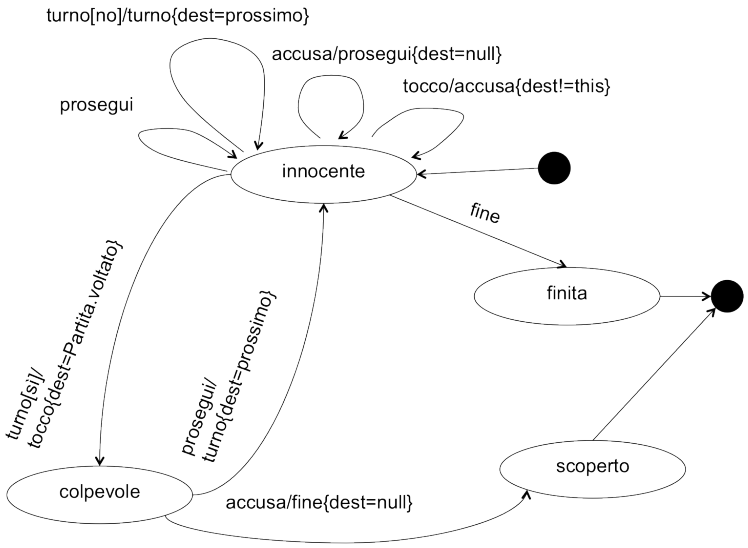
Analisi

Specifica del diagramma degli stati e delle transizioni della classe *Soldato*

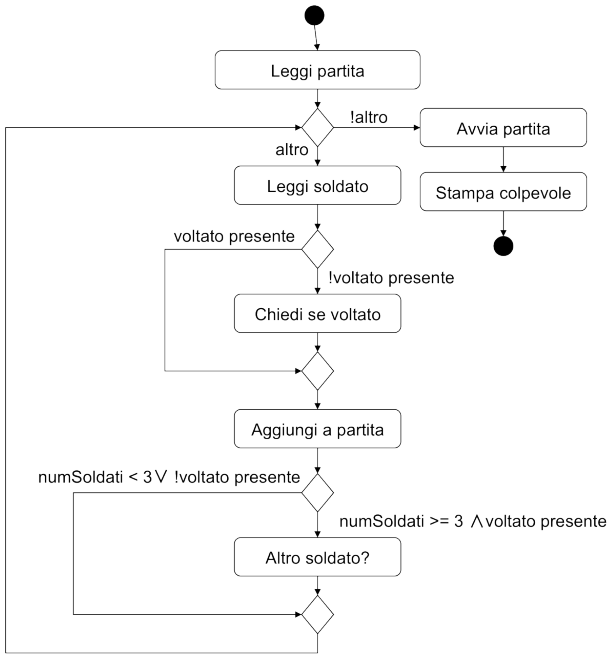
```
InizioSpecificaStatiClasse Soldato
    Stato: {innocente, colpevole, scoperto, finita}
```



(a) Diagramma UML delle classi



(b) Diagramma UML degli stati e delle transizioni



(c) Diagramma UML delle attività

```
    Variabili di stato ausiliarie:  --
    Stato iniziale:
       stato = innocente
FineSpecifica

InizioSpecificaTransizioniClasse Soldato
    Transizione: innocente -> innocente
                      prosegui
        Evento: prosegui
        Condizione: --
        Azione: --


    Transizione: innocente -> innocente
                    turno[no] / turno{dest = prossimo}
        Evento: turno
        Condizione: [no]: non si vuole toccare il soldato voltato (decisione casuale)
        Azione:
           pre: evento.dest = this
           post: nuovoevento = passa{dest = prossimo}
                dove prossimo è il primo soldato non voltato successivo a quello corrente, nella partita


    Transizione: innocente -> innocente
                    accusa / prosegui{dest = null}
        Evento: accusa
        Condizione:--
        Azione:
           pre: evento.dest = this
           post: nuovoevento = prosegui{dest = null}

    Transizione: innocente -> innocente
                    tocco / accusa{dest != this}
        Evento: tocco
        Condizione:--
        Azione:
           pre: evento.dest = this
           post: nuovoevento = accusa{dest != this}

    Transizione: innocente -> colpevole
                    turno[sì] / tocco{dest = partita.voltato}
        Evento: turno
        Condizione: [sì]: si vuole toccare il soldato voltato (decisione casuale)
        Azione:
           pre: evento.dest = this
           post: nuovoevento = tocco{dest = partita.voltato}
                dove partita.voltato è il soldato che, nella partita corrente, gioca nel ruolo voltato


    Transizione: innocente -> finita
                         fine
        Evento: fine
        Condizione:--
        Azione: --


    Transizione: colpevole -> innocente
                    prosegui / turno{dest = prossimo}
        Evento: prosegui
        Condizione:--
        Azione:
           pre: evento.dest = this
           post: nuovoevento = turno{dest = prossimo}
                dove prossimo è il primo soldato non voltato successivo a quello corrente, nella partita


    Transizione: colpevole -> scoperto
                    accusa / fine{dest = null}
```

Evento: accusa
Condizione:--
Azione:
pre: evento.dest = this
post: nuovoevento = fine{dest = null}

FineSpecifica

Attività di I/O

InizioSpecificaAttivitàAtomica LeggiPartita
LeggiPartita ():(Partita)
pre: --
post: Mostra all'utente la finestra per l'inserimento del nome della nuova partita.
result è l'oggetto di classe *Partita* avente il nome specificato dall'utente.
FineSpecifica

InizioSpecificaAttivitàAtomica LeggiSoldato
LeggiSoldato ():(Soldato)
pre: --
post: Mostra all'utente la finestra per l'inserimento dei dati dal nuovo soldato da creare.
result è l'oggetto di classe *Soldato* creato usando i dati inseriti dall'utente.
FineSpecifica

InizioSpecificaAttivitàAtomica ChiediSeVoltato
ChiediSeVoltato ():(Boolean)
pre: --
post: Mostra all'utente una finestra da cui è possibile indicare se l'ultimo soldato creato debba giocare nel ruolo *voltato* o meno.
result è `true` se e solo se il soldato deve giocare nel ruolo *voltato*.
FineSpecifica

InizioSpecificaAttivitàAtomica ChiediSeAltroSoldato
ChiediSeAltroSoldato ():(Boolean)
pre: --
post: Mostra all'utente una finestra è possibile indicare se si abbia intenzione di inserire un nuovo soldato o meno.
result è `true` se e solo se si vuole inserire un nuovo soldato.
FineSpecifica

InizioSpecificaAttivitàAtomica StampaColpevole
StampaColpevole (Partita p):()
pre: --
post: Stampa i dati del soldato che, nella partita `p`, si trova nello stato `scoperto`
FineSpecifica

Attività Atomiche

InizioSpecificaAttivitàAtomica AggiungiAPartita
AggiungiAPartita(Partita p, Soldato s, Boolean voltato):()
pre: --
post: Inserisce un link `partecipa` tra `p` ed `s`. Se `voltato=true` allora inserisce anche un link di tipo `voltato` tra `p` ed `s`.
FineSpecifica

InizioSpecificaAttivitàAtomica AvviaPartita
AvviaPartita (Partita p):()
pre: --
post: Invia l'evento `turno` al primo soldato non voltato che partecipa alla partita `p`.

FineSpecifica

NOTA: L'attività atomica *AttendiTerminePartita*, presente nel codice, è usata solo per ragioni implementative, e può essere trascurata in questa sede.

Attività Composte

InizioSpecificaAttività AttivitaPrincipale

```
    AttivitaPrincipale():()
        Variabili Processo:
            altroSoldato: Boolean; // true sse si vuiole/deve aggiungere un altro soldato alla partita
            voltatoPresente: Boolean; // true sse esiste un soldato voltato nella partita corrente
            numSoldati : Int; // numero di soldati nella partita corrente
            partitaCorrente: Partita;
            soldatoCorrente: Soldato;
            soldatoCorrenteVoltato: Boolean; // true sse il soldato corrente gioca nel ruolo voltato

        Inizio Processo
            altroSoldato := true;
            voltatoPresente := false;
            numSoldati := 0;

            LeggiPartita():(partitaCorrente);

            while (altroSoldato) {
                soldatoCorrenteVoltato := false;
                LeggiSoldato():(soldatoCorrente);
                if (!voltatoPresente){
                    ChiediSeVoltato():(soldatoCorrenteVoltato);
                    if (soldatoCorrenteVoltato){
                        voltatoPresente := true;
                    }
                }

                AggiungiAPartita(partitaCorrente, soldatoCorrente, soldatoCorrenteVoltato);
                numSoldati++;

                if (numSoldati >= 3 && voltatoPresente){
                    ChiediSeAltroSoldato():(altroSoldato);
                }
            }

            AvviaPartita(partitaCorrente):();
            StampaColpevole(partitaCorrente):();
```

FineSpecifica

Progetto

Responsabilità sulle Associazioni

R: Requisiti; O: Specifica delle Operazioni/Attività; M: Vincoli di Molteplicità

Associazione	Classe	Ha Responsabilità
voltato	Soldato	NO (O)
	Partita	SÌ (M)
partecipa	Soldato	SÌ (M,O)
	Partita	SÌ (M)

NOTA: la responsabilità di *Soldato* verso *partecipa* derivante dalle operazioni (O), è dovuta al diagramma degli stati e delle transizioni. Infatti *Soldato* deve conoscere: i. l'ordine dei soldati, per passare l'evento *turno* al suo successore; ii. il soldato voltato, per potergli passare l'evento *tocco*.

Strutture di Dati

Rappresentiamo le collezioni omogenee di oggetti mediante le classi `Set` ed `HashSet` del Collection Framework di Java.

Inoltre, rappresentiamo le collezioni omogenee ordinate di oggetti mediante le classi `List` e `LinkedList` del Collection Framework di Java.

Tabelle di Gestione delle Proprietà delle Classi UML

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

Classe UML	Proprietà Immutabile
Soldato	nome
	cognome
	matricola
Partita	nome

	Proprietà	
Classe UML	**Nota alla nascita**	**Non nota alla nascita**
-	-	-

Altre Considerazioni

Non dobbiamo assumere una particolare sequenza di nascita degli oggetti.

Non esistono valori di default per qualche proprietà che siano validi per tutti gli oggetti.