Sapienza Università di Roma
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Ingegneria Informatica ed Automatica, Ingegneria dei Sistemi Informatici

Esercitazioni di Progettazione del Software
A.A. 2013/2014

Prova al calcolatore – 20 giugno 2014

Requisiti

Si vuole realizzare un'applicazione per la simulazione di gare di salto con lo snowboard. Una gara è caratterizzata da un codice (una stringa) e dalla data in cui si svolge. Ad una gara partecipano almeno due e non più di sei atleti. Ogni atleta, di cui interessano nome, cognome e nazione di appartenenza, partecipa ad una gara alla volta, effettuando un salto di una certa lunghezza (in metri). Una gara avviene su un certo percorso, caratterizzato dalla lunghezza minima del salto da effettuare e dall'altezza da cui avviene il salto (entrambe in metri). Ogni atleta possiede una tavola da snowboard, di cui interessa la marca. Tra gli atleti partecipanti ad una gara, risultano vincitori (è considerato anche il pari merito) coloro che effettuano il salto più lungo.

In Figura 1 è mostrato il diagramma delle classi corrispondente al dominio.

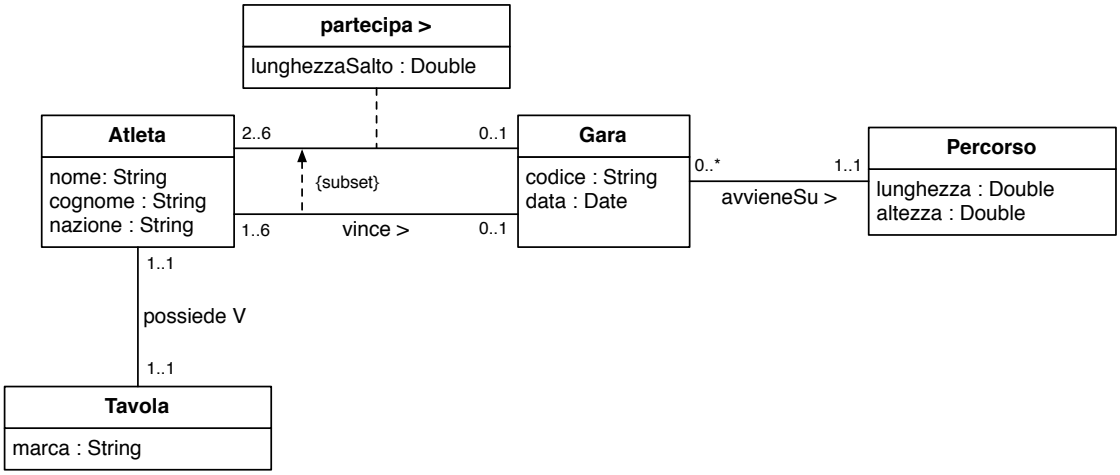

Figura 1: Diagramma UML delle classi

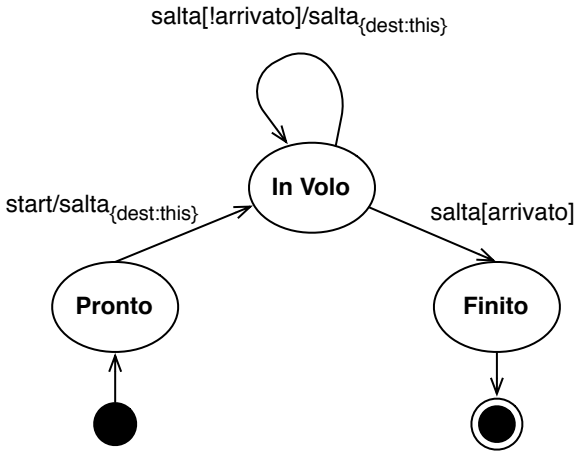

Figura 2: Diagramma degli stati e delle transizioni relativo alla classe *Atleta*

Una gara si svolge come segue. Ogni atleta è inizialmente nello stato *Pronto*, in attesa del segnale di partenza (evento *start* generato dall'utente). Quando riceve il segnale, l'atleta passa nello stato $In\ Volo$ ed effettua il salto, inviando a se stesso l'evento *salta*.

Durante la gara, ciascun atleta che si trova $In\ Volo$ si comporta come segue alla ricezione di un evento *salta*:

- se l'atleta ha completato il suo salto e tocca terra (condizione *arrivato*), passa nello stato *Finito*;
- altrimenti, l'atleta avanza in aria durante il salto aggiornando la lunghezza del salto stesso e invia a sé l'evento *salta*.

L'avanzamento degli atleti durante il salto avviene secondo una funzione i cui dettagli non sono qui specificati, così come le condizioni per determinare se l'atleta ha completato il salto.[1]

In Figura 2 è mostrato il diagramma degli stati e delle transizioni relativo alla classe *Atleta*.

Una sessione di interazione con l'applicazione si svolge come segue:

- il sistema genera una nuova gara;
- l'utente inserisce i dati del percorso di salto, specificandone la lunghezza e l'altezza;
- il percorso viene associato alla gara da svolgere;
- iterativamente vengono definiti gli atleti che partecipano alla gara; in particolare:
  - l'utente inserisce i dati di un atleta, specificandone il nome, cognome e nazione di appartenenza;
  - l'utente specifica poi la marca della tavola posseduta dall'atleta, e la tavola viene associata all'atleta;
  - l'atleta viene iscritto come partecipante alla gara e si procede poi con l'eventuale atleta successivo, sulla base della scelta dell'utente;

[1] i dettagli non sono rilevanti ai fini della prova e le corrispondenti funzioni sono date.

- dopo aver definito gli atleti che partecipano alla gara, si verifica la regolarità della gara, controllando che il numero di partecipanti sia conforme ai limiti che prevedono un minimo di due ed un massimo di 6 partecipanti;
- se la gara non è regolare, viene visualizzato un messaggio di errore e l'applicazione termina;
- se la gara è regolare, la simulazione di gara di salto viene inizializzata e poi visualizzata tramite opportuna interfaccia grafica che mostra il salto degli atleti;
- al termine della gara si determinano i vincitori della gara di salto e viene visualizzata una schermata riassuntiva che riporta i vincitori e la lunghezza del salto effettuato da ogni atleta.

In Figura 3 è riportato il diagramma delle attività corrispondente.

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[2], si chiede di intervenire sulle seguenti classi:

- `FinestraPrincipale` (package `app.gui`) (si vedano le considerazioni in fondo al documento di specifica)
- `Gara` (package `app.dominio`)
- `ManagerVince` (package `app.dominio`)
- `AtletaFired` (package `app.dominio`)
- `RegistraPercorso` (package `app.attivita.atomiche`)
- `AssegnaTavola` (package `app.attivita.atomiche`)
- `AttivitaPrincipale` (package `app.attivita.complesse`)

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.

[2] le porzioni di codice su cui intervenire sono identificate dal commento `/* DA COMPLETARE A CURA DELLO STUDENTE */`

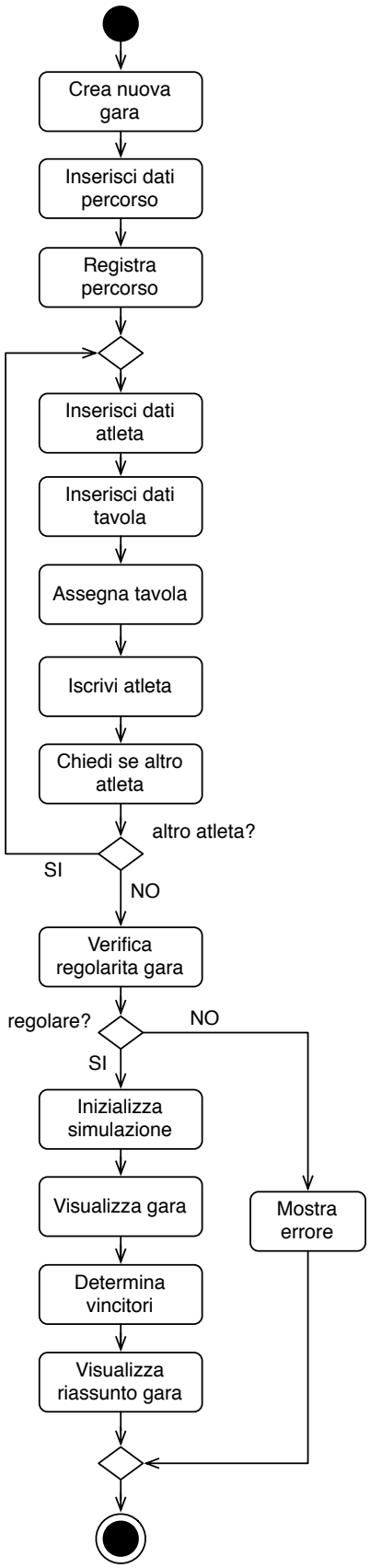

Figura 3: Diagramma delle attività

Analisi

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

InizioSpecificaStatiClasse Atleta

Stato: {pronto, inVolo, finito}
Variabili di stato ausiliarie: parametri che determinano la posizione e il percorso di salto[3]
Stato iniziale:
stato = pronto

FineSpecifica

InizioSpecificaTransizioniClasse Atleta

Transizione: pronto → inVolo
start/salta{dest:this}
Evento: start
Condizione: --
Azione:
pre: --
post: nuovoevento = salta{mitt=this, dest=this}

Transizione: inVolo → inVolo
salta[!arrivato]/salta{dest:this}
Evento: salta
Condizione: ([!arrivato])
NOTA: si veda il metodo ausiliario `arrivato()` nella classe *AtletaFired*
Azione:
pre: --
post: nuovoevento = salta{mitt=this, dest=this} AND
sia *link* il link di tipo *Partecipa* tale che *link.atleta* = `this`; allora rimpiazza il link esistente con un nuovo link *link′* tale che *link′.mtPercorsi* riporti la lunghezza corrente del salto.
NOTA: si veda il metodo ausiliario `aggiornaMtPercorsi()` nella classe *AtletaFired*

Transizione: inVolo → finito
salta[arrivato]
Evento: salta
Condizione: ([arrivato])
NOTA: si veda il metodo ausiliario `arrivato()` nella classe *AtletaFired*
Azione: --

FineSpecifica

[3] si veda la classe `AtletaFired` per i dettagli

Attività di I/O

`InizioSpecificaAttivitàAtomica InserisciDatiPercorso`
`InserisciDatiPercorso ():(Percorso)`
`pre: --`
`post:` Legge lunghezza e altezza del percorso di salto, forniti in input dall'utente.
`result` è il percorso creata a partire dai dati inseriti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica InserisciDatiAtleta`
`InserisciDatiAtleta ():(Atleta)`
`pre: --`
`post:` Legge il nome, il cognome e la nazione di appartenenza di un atleta, forniti in input dall'utente.
`result` è l'atleta creato a partire dai dati inseriti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica InserisciDatiTavola`
`InserisciDatiTavola ():(Tavola)`
`pre: --`
`post:` Consente all'utente di selezionare una marca per una tavola da snowboard.
`result` è la tavola creata a partire dai dati inseriti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica ChiediSeAltroAtleta`
`ChiediSeAltroAtleta ():(Bool)`
`pre: --`
`post:` Chiede all'utente se vuole iscrivere un altro atleta alla gara.
`result` è `true` in caso affermativo, `false` altrimenti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica MostraErrore`
`MostraErrore ():()`
`pre: --`
`post:` Visualizza un messaggio di errore che informa l'utente della non regolarità della gara.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica VisualizzaGara`
`VisualizzaGara (g:Gara):()`
`pre: --`
`post:` Mostra una finestra di visualizzazione della gara `g`.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica VisualizzaRiassuntoGara`
`VisualizzaRiassuntoGara (g:Gara):()`
`pre: --`
`post:` Visualizza i metri percorsi in salto da ciascun atleta che ha partecipato alla gara `g` e stampa i nomi degli atleti vincitori della gara di salto.
`FineSpecifica`

Attività Atomiche

`InizioSpecificaAttivitàAtomica CreaNuovaGara`
`CreaNuovaGara ():(Gara)`
`pre: --`
`post:` Crea una nuova gara con codice e data generati dal sistema
`result` è la gara creata.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica RegistraPercorso`
`RegistraPercorso (g:Gara, p:Percorso) : ()`
`pre: --`
`post:` Crea un link di tipo *AvvieneSu* tra la gara `g` e il percorso `p`.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica AssegnaTavola`
`AssegnaTavola (a:Atleta, t:Tavola) : ()`
`pre: --`
`post:` Crea un link di tipo *Possiede* tra l'atleta `a` e la tavola `t`.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica IscriviAtleta`
`IscriviAtleta (g:Gara, a:Atleta) : ()`
`pre: --`
`post:` Crea un link *link* di tipo *Partecipa* tra la gara `g` e l'atleta `a`, tale che $link.lunghezzaSalto = 0$.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica VerificaRegolaritaGara`
`VerificaRegolaritaGara (g:Gara) : (Bool)`
`pre: --`
`post:` Sia P l'insieme dei link di tipo *Partecipa* che coinvolgono la gara `g`.
`result` è `true` se $|P| \geq 2 \wedge |P| \leq 6$, `false` altrimenti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica InizializzaSimulazione`
`InizializzaSimulazione(g:Gara):()`
`pre: --`
`post:` Inizializza l'`Environment`, inserendovi (come *Listener*) tutti gli atleti legati alla gara `g` da un link di tipo *Partecipa*; successivamente, attiva i *Listener*.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica DeterminaVincitori`
`DeterminaVincitori(g:Gara):()`
`pre: --`
`post:` Crea un link di tipo $Vince$ tra la gara `g` e ogni atleta che ha partecipato alla gara e ha effettuato il salto più lungo.
In particolare, sia P l'insieme dei link di tipo *Partecipa* che coinvolgono la gara `g`; per ogni link $l \in P$, se $l.lunghezzaSalto = \max\{m.lunghezzaSalto \mid m \in P\}$ allora crea un link di tipo $Vince$ tra l'atleta $l.atleta$ e la gara `g`.
`FineSpecifica`

Attività Composte

InizioSpecificaAttività AttivitaPrincipale

AttivitaPrincipale():()

Variabili Processo:
gara: Gara -- la gara
percorso: Percorso -- il percorso
atleta: Atleta -- atleta corrente
tavola: Tavola -- tavola corrente
altroAtleta: Bool -- altro atleta da aggiungere?
garaRegolare: Bool -- gara regolare?

Inizio Processo:

```
CreaNuovaGara():(gara);
InserisciDatiPercorso():(percorso);
RegistraPercorso(gara, percorso):();

do {
  InserisciDatiAtleta():(atleta);
  InserisciDatiTavola():(tavola);
  AssegnaTavola(atleta, tavola):();
  IscriviAtleta(gara, atleta):();
  ChiediSeAltroAtleta():(altroAtleta);
} while(altroAtleta);

VerificaRegolaritaGara(gara):(garaRegolare);

if (garaRegolare) {
  InizializzaSimulazione(gara):();
  VisualizzaGara(gara):();
  DeterminaVincitori(gara):();
  VisualizzaRiassuntoGara(gara):();
}
else {
  MostraErrore():();
}
```

FineSpecifica

Progetto

Responsabilità sulle Associazioni

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

Associazione	Classe	Ha Responsabilità
partecipa	Atleta	SI (R, M, O)
	Gara	SI (R, M, O)
vince	Atleta	SI (M)
	Gara	SI (R, M, O)
possiede	Atleta	SI (R, M, O)
	Tavola	SI (M)
avvieneSu	Gara	SI (R, M, O)
	Percorso	NO

Strutture di Dati

Rappresentiamo le collezioni omogenee di oggetti mediante le classi `Set` ed `HashSet` 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.

Classe UML	Proprietà Immutabile
Gara	codice data
Atleta	nome cognome
Tavola	marca
Percorso	lunghezza altezza

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.

La finestra principale dell'applicazione deve essere simile a quella in Figura 4.

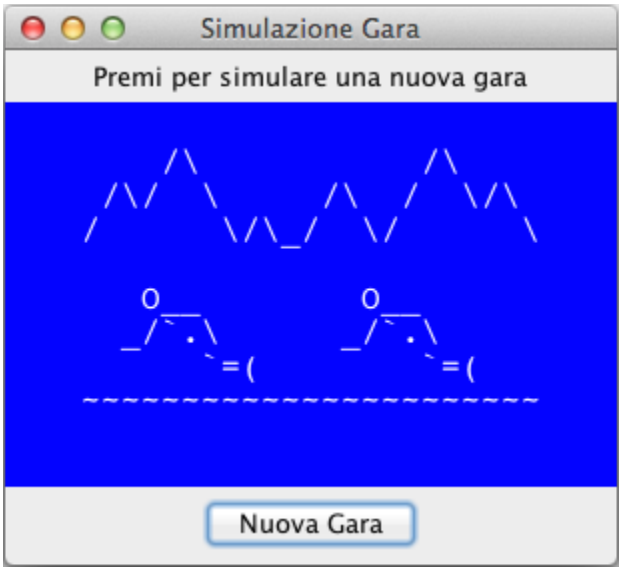

Figura 4: La finestra principale