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. 2012/2013

# Prova al calcolatore – 11 febbraio 2014

## Requisiti

Si vuole realizzare un'applicazione per la simulazione di regate di canottaggio. Una regata è caratterizzata dal nome (stringa) e dalla distanza partenza-traguardo, misurata in chilometri. Ad una regata partecipano almeno due equipaggi, ciascuno caratterizzato dal nome. Un equipaggio può partecipare ad al più una regata per volta. Per ogni regata e per ciascun equipaggio che vi partecipa, è di interesse conoscere quanti chilometri ha percorso l'equipaggio nella regata. Tra gli equipaggi che partecipano ad una regata, alcuni sono vincitori (è considerato anche il pari merito), stabiliti al termine della regata in base al numero di chilometri percorsi. Ogni regata ha almeno un vincitore.

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

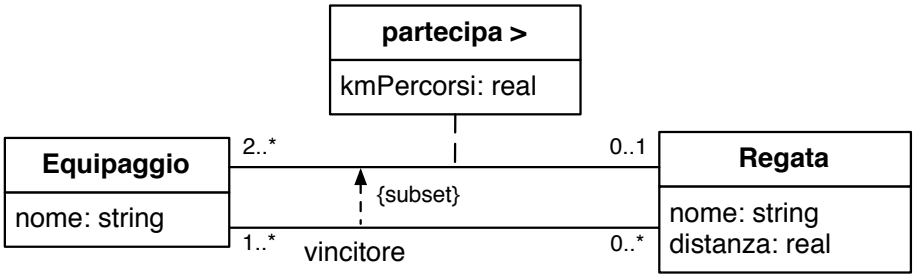


Figura 1: Diagramma UML delle classi

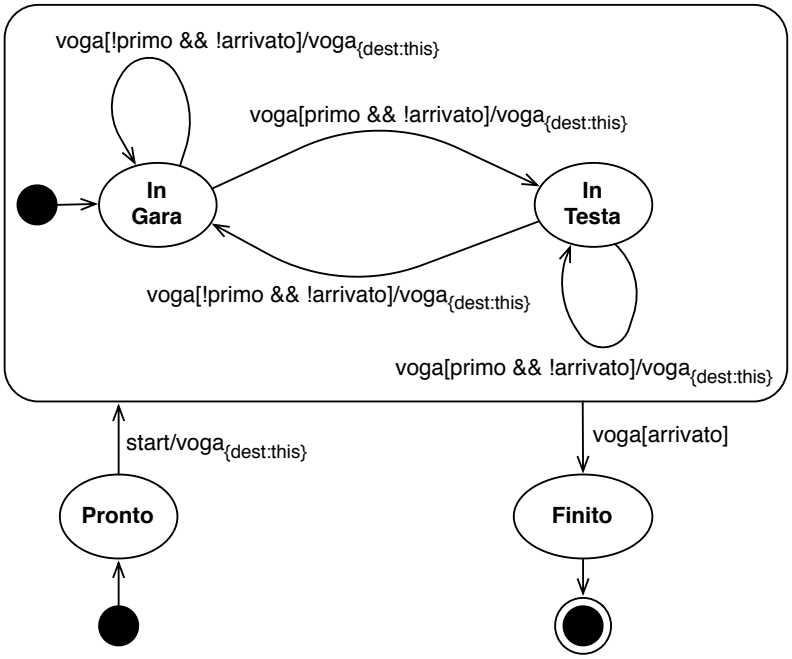


Figura 2: Diagramma degli stati e delle transizioni relativo alla classe *Equipaggio*

Una regata si svolge come segue. Ogni equipaggio è inizialmente nello stato *Pronto*, in attesa del segnale di partenza (evento *start* generato dall'utente). Quando riceve il segnale, l'equipaggio passa nello stato *InGara* ed inizia a vogare, inviando a se stesso l'evento *voga*.

Durante la regata, *finché nessun equipaggio ha raggiunto o superato il traguardo* (condizione !*arrivato*), ciascun equipaggio si comporta come segue alla ricezione di un evento *voga*:

- quando è nello stato *InGara* o *InTesta*, se è tra i partecipanti che hanno percorso il maggior numero di chilometri (condizione *primo*) – possono essercene diversi a pari merito – allora passa (o rimane) nello stato *InTesta*, avanza sul campo di regata (aggiornando i chilometri percorsi) e invia a se stesso l'evento *voga*;
- quando è nello stato *InGara* o *InTesta*, se *non* è tra i partecipanti che hanno percorso il maggior numero di chilometri (condizione !*primo*) allora passa (o rimane) nello stato *InGara*, avanza sul campo di regata (aggiornando i chilometri percorsi) e invia a se stesso l'evento *voga*.

Da entrambi gli stati *InGara* e *InTesta*, quando un equipaggio (incluso esso stesso) raggiunge o supera il traguardo (condizione *arrivato*), entra nello stato $finito$.

Gli equipaggi, ad ogni passo, avanzano con un ritmo che dipende dallo stato in cui si trovano:

- nello stato *InGara*, un equipaggio avanza di una distanza $d = l/100 \cdot (1 + r)$, dove $l$ è la distanza partenza-traguardo della regata, ed $r$ un valore reale casuale nell'intervallo $[0, 1)$;
- nello stato *InTesta*, un equipaggio avanza di una distanza $d = l/100 \cdot (1.1 - r)$, con $l$ ed $r$ definiti come al punto precedente (si assume che quando è in testa, l'equipaggio procede più lentamente perché non ha come riferimento alcun equipaggio che lo precede).

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

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

- l'utente inserisce i dati della regata, specificandone il nome e la distanza partenza-traguardo;
- l'utente inserisce i dati di un equipaggio, specificandone il nome;
- l'equipaggio viene iscritto come partecipante alla regata;
- iterativamente vengono definiti gli ulteriori equipaggi che partecipano alla regata; in particolare:
  - l'utente inserisce i dati di un equipaggio, specificandone il nome;
  - l'equipaggio viene iscritto come partecipante alla regata;
  - si procede poi con l'eventuale equipaggio successivo, sulla base della scelta dell'utente;
- dopo aver definito gli equipaggi che partecipano alla regata, la simulazione di regata viene inizializzata e poi visualizzata tramite opportuna interfaccia grafica che mostra il percorso degli equipaggi e consente all'utente di generare gli eventi per controllare la regata;
- al termine della gara si determinano i vincitori della regata e viene visualizzata una schermata riassuntiva che riporta i vincitori ed i chilometri percorsi da ogni equipaggio.

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

- `FinestraPrincipale` (package `app.gui`) (si vedano le considerazioni in fondo al documento di specifica)
- `Regata` (package `app.dominio`)
- `EquipaggioFired` (package `app.dominio`)
- `IscriviEquipaggio` (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.

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

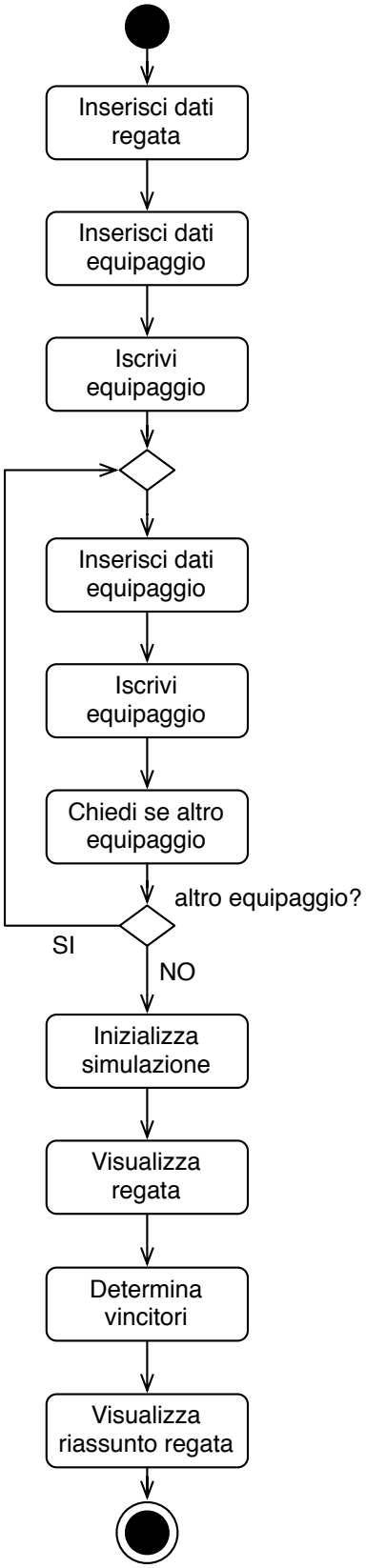


Figura 3: Diagramma delle attività

# Analisi

## Specifica del diagramma degli stati e delle transizioni della classe *Equipaggio*

InizioSpecificaStatiClasse Equipaggio
Stato: {pronto, inGara, inTesta, finito}
Variabili di stato ausiliarie: –
Stato iniziale:
stato = pronto
FineSpecifica

InizioSpecificaTransizioniClasse Equipaggio
Transizione: pronto → inGara
start/voga{dest:this}
Evento: start
Condizione: --
Azione:
pre: --
post: nuovoevento = voga{mitt=this, dest=this}

Transizione: inGara → inGara
voga[!primo∧!arrivato]/voga{dest:this}
Evento: voga
Condizione: ([!primo∧!arrivato])
Sia *link* il link di tipo *Partecipa* tra l'equipaggio `this` e la regata $rg$, e sia $d = link.kmPercorsi$ la distanza percorsa dall'equipaggio. Sia inoltre $P$ l'insieme dei link di tipo *Partecipa* che coinvolgono la regata $rg$.

Condizione `!arrivato`
L'equipaggio non è arrivato se tutti gli equipaggi che partecipano alla regata hanno percorso una distanza inferiore alla distanza totale della regata, cioè
$\forall l' \in P \rightarrow l'.kmPercorsi < rg.distanza$
**NOTA:** si veda il metodo ausiliario `arrivato()` nella classe *EquipaggioFired*

Condizione `!primo`
L'equipaggio non è primo se esiste un altro equipaggio che partecipa alla regata che ha percorso una distanza maggiore, cioè
$\exists l' \in P \mid l'.equipaggio \neq$ `this` $\wedge\, l'.kmPercorsi > link.kmPercorsi$
**NOTA:** si veda il metodo ausiliario `primo()` nella classe *EquipaggioFired*
Azione:
pre: --
post: nuovoevento = voga{mitt=this, dest=this} AND
sia *link* il link di tipo *Partecipa* tale che $link.equipaggio =$ `this`; allora rimpiazza il link esistente con un nuovo link $link'$ tale che
$link'.kmPercorsi = link.kmPercorsi + (link.regata.distanza/100) \cdot (1 + rand([0, 1)))$.
**NOTA:** si veda il metodo ausiliario `aggiornaKmPercorsi()` nella classe *EquipaggioFired*

Transizione: `inTesta` → `inTesta`
`voga[primo∧!arrivato]/voga{dest:this}`

Evento: `voga`

Condizione: (`[primo∧!arrivato]`)

Sia *link* il link di tipo *Partecipa* tra l'equipaggio `this` e la regata $rg$, e sia $d = link.kmPercorsi$ la distanza percorsa dall'equipaggio. Sia inoltre $P$ l'insieme dei link di tipo *Partecipa* che coinvolgono la regata $rg$.

Condizione `!arrivato`

-- Analoga alla condizione di `inGara` → `inGara`

Condizione `primo`

L'equipaggio è primo se tutti gli altri equipaggi che partecipano alla regata hanno percorso una distanza minore o uguale alla distanza percorsa dall'equipaggio `this`, cioè
$\forall l' \in P \mid (l'.equipaggio \neq \texttt{this}) \rightarrow l'.kmPercorsi \leq link.kmPercorsi$
**NOTA:** si veda il metodo ausiliario `primo()` nella classe *EquipaggioFired*

Azione:

pre: --

post: `nuovoevento = voga{mitt=this, dest=this} AND`
sia *link* il link di tipo *Partecipa* tale che $link.equipaggio = \texttt{this}$; allora rimpiazza il link esistente con un nuovo link $link'$ tale che
$link'.kmPercorsi = link.kmPercorsi + (link.regata.distanza/100) \cdot (1.1 - rand([0,1)))$.
**NOTA:** si veda il metodo ausiliario `aggiornaKmPercorsi()` nella classe *EquipaggioFired*

Transizione: `inGara` → `inTesta`
`voga[primo∧!arrivato]/voga{dest:this}`

Evento: `voga`

Condizione: (`[primo∧!arrivato]`)

-- Analoga alla condizione di `inTesta` → `inTesta`

Azione:

-- Analoga all'azione di `inGara` → `inGara`

Transizione: `inTesta` → `inGara`
`voga[!primo∧!arrivato]/voga{dest:this}`

Evento: `voga`

Condizione: (`[!primo∧!arrivato]`)

-- Analoga alla condizione di `inGara` → `inGara`

Azione:

-- Analoga all'azione di `inTesta` → `inTesta`

Transizione: `inGara` → `finito`, `inTesta` → `finito`
`voga[arrivato]`

Evento: `voga`

Condizione: (`[arrivato]`)

Sia $rg$ la regata legata all'equipaggio `this` da un link di tipo *Partecipa*. Sia inoltre $P$ l'insieme dei link di tipo *Partecipa* che coinvolgono la regata $rg$.
L'equipaggio è arrivato se esiste un equipaggio che partecipa alla regata che abbia percorso una distanza uguale o superiore alla distanza totale della regata, cioè
$\exists l' \in P \mid l'.kmPercorsi \geq rg.distanza$

Azione:

pre: --

post: --

FineSpecifica

## Attività di I/O

`InizioSpecificaAttivitàAtomica` `InserisciDatiRegata`
`InserisciDatiRegata ():(Regata)`
`pre: --`
`post:` Legge nome e distanza partenza-traguardo di una regata, forniti in input dall'utente.
`result` è la regata creata a partire dai dati inseriti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica` `InserisciDatiEquipaggio`
`InserisciDatiEquipaggio ():(Equipaggio)`
`pre: --`
`post:` Legge il nome dell'equipaggio, fornito in input dall'utente.
`result` è l'equipaggio creato a partire dai dati inseriti.
`FineSpecifica`

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

`InizioSpecificaAttivitàAtomica` `VisualizzaRegata`
`VisualizzaRegata (r:Regata):()`
`pre: --`
`post:` Mostra la finestra di visualizzazione della regata `r`.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica` `VisualizzaRiassuntoRegata`
`VisualizzaRiassuntoRegata (r:Regata):()`
`pre: --`
`post:` Visualizza i chilometri percorsi da ciascun equipaggio nella regata `r` e stampa i nomi degli equipaggi vincitori della regata.
`FineSpecifica`

## Attività Atomiche

`InizioSpecificaAttivitàAtomica` `IscriviEquipaggio`
`IscriviEquipaggio (r:Regata, e:Equipaggio) : ()`
`pre: --`
`post:` Crea un link *link* di tipo *Partecipa* tale che $link.kmPercorsi = 0$, $link.regata =$ `r`, $link.equipaggio =$ `e`
`FineSpecifica`

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

InizioSpecificaAttivitàAtomica DeterminaVincitori

DeterminaVincitori(r:Regata):()

pre: --

post: Crea un link di tipo $Vincitore$ tra la regata `r` e ogni equipaggio che ha partecipato alla regata e ha percorso il maggior numero di chilometri.
In particolare, sia $P$ l'insieme dei link di tipo $Partecipa$ che coinvolgono la regata `r`; per ogni link $l \in P$, se $l.kmPercorsi = \max\{m.kmPercorsi \mid m \in P\}$ allora crea un link di tipo $Vincitore$ tra l'equipaggio $l.equipaggio$ e la regata `r`.

FineSpecifica

## Attività Composte

InizioSpecificaAttività AttivitaPrincipale

AttivitaPrincipale():()

Variabili Processo:

`regata: Regata` -- la regata
`equipaggio: Equipaggio` -- equipaggio corrente
`altroEquipaggio: Bool` -- altro equipaggio da aggiungere?

Inizio Processo:

```
InserisciDatiRegata():(regata);

InserisciDatiEquipaggio():(equipaggio);

IscriviEquipaggio(regata, equipaggio):();

do {
  InserisciDatiEquipaggio():(equipaggio);
  IscriviEquipaggio(regata, equipaggio):();
  ChiediSeAltroEquipaggio():(altroEquipaggio);
} while(altroEquipaggio);

InizializzaSimulazione(regata):();

VisualizzaRegata(regata):();

DeterminaVincitori(regata):();

VisualizzaRiassuntoRegata(regata):();
```

FineSpecifica

# Progetto

## Responsabilità sulle Associazioni

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

| Associazione | Classe | Ha Responsabilità |
|---|---|---|
| partecipa | Equipaggio<br>Regata | SI (M,O,R)<br>SI (M,O) |
| vincitore | Equipaggio<br>Regata | NO<br>SI (M,O) |

## 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 |
|---|---|
| Regata | nome distanza |
| Equipaggio | nome |

## 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.

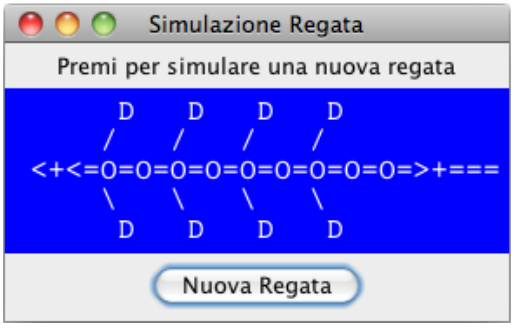


Figura 4: La finestra principale