

Esercitazioni di Progettazione del Software - A.A. 2009/2010
Prova al calcolatore del 18 giugno 2010

Requisiti

L'applicazione da realizzare riguarda la simulazione di una gara ciclistica. Ogni gara è caratterizzata dal nome (stringa) e dalla distanza partenza-traguardo, misurata in chilometri (reale). Ad una gara partecipano almeno due ciclisti, ciascuno caratterizzato dal proprio nome. Un ciclista può partecipare ad al più una gara per volta. Per ogni gara e per ciascun ciclista che vi partecipa, è d'interesse conoscere quanti chilometri il ciclista ha percorso nella gara. Tra i ciclisti che partecipano ad una gara, alcuni sono vincitori (è considerato anche il pari merito), stabiliti al termine della gara in base al numero di chilometri percorsi. Ogni gara ha almeno un vincitore. In Figura 1(a) è mostrato il diagramma delle classi corrispondente al dominio.

La gara si svolge come segue. I ciclisti sono inizialmente nello stato *pronto*, in attesa del segnale di partenza (evento *start*). Quando ricevono il segnale, entrano nello stato *inGara* ed iniziano a correre (evento *corri*). Durante la gara, *finché nessun partecipante ha raggiunto o superato il traguardo* (condizione *!arrivato*), ciascun ciclista si comporta come segue:

- 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 entra (o rimane) nello stato *inTesta*;
- quando è nello stato *inGara* o *inTesta*, se *non* è tra i partecipanti che hanno percorso il maggior numero di chilometri (condizione *!primo*) allora entra (o rimane) nello stato *inGara*;
- da entrambi gli stati *inGara* e *inTesta*, quando qualcuno (incluso egli stesso) raggiunge o supera il traguardo (condizione *arrivato*), entra nello stato *finito*.

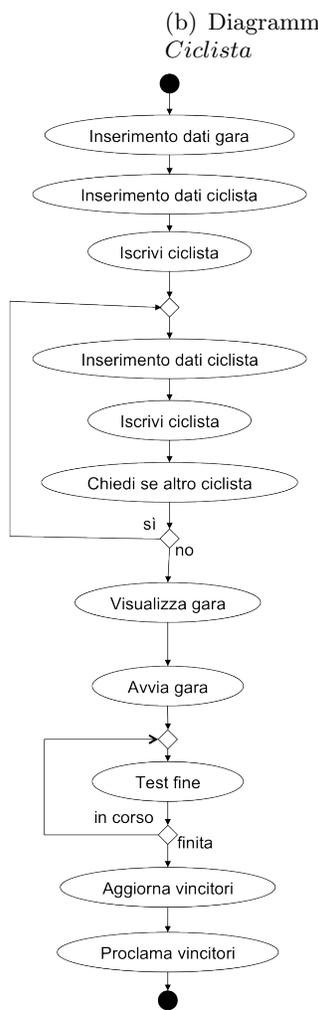
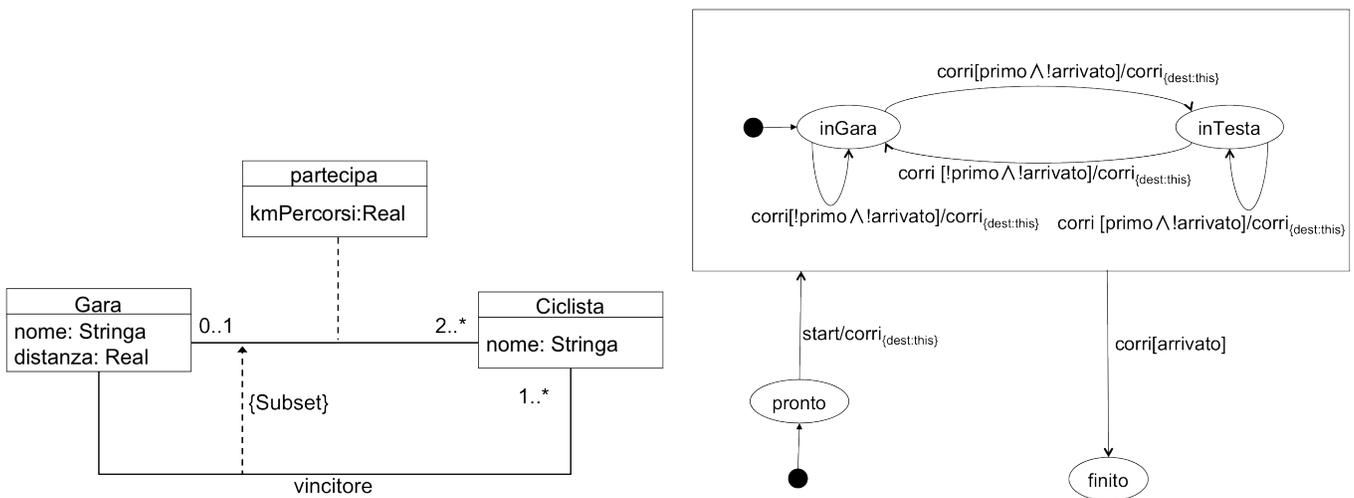
I ciclisti, ad ogni passo, avanzano con un ritmo che dipende dallo stato in cui si trovano:

- nello stato *inGara*, un ciclista avanza di una distanza $d = l/100 \cdot (1 + r)$, dove l è la distanza partenza-traguardo della gara, ed r un valore reale casuale nell'intervallo $[0, 1)$; ¹
- nello stato *inTesta*, un ciclista avanza di una distanza $d = l/100 \cdot (1.1 - r)$, con l ed r definiti come al punto precedente (quando è in testa, il ciclista procede più lentamente perché non può seguire la scia del ciclista che lo precede).

In Figura 1(b) è mostrato il diagramma degli stati e delle transizioni relativo alla classe *Ciclista*.

L'applicazione deve:

¹Si ricorda che il metodo statico `random()` della classe `Math` restituisce un `double` scelto casualmente nell'intervallo $[0, 1)$.



- creare una nuova gara a partire da dati (nome e distanza partenza-traguardo) forniti in input;
- creare i ciclisti (almeno 2) ed iscriverli alla gara, a partire da dati (nome) forniti in input;
- simulare la gara, visualizzandone l'evoluzione mediante opportuna interfaccia grafica;
- al termine della simulazione, aggiornare i vincitori della gara e stamparne i nomi.

In Figura 1(c) 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, si chiede di intervenire sulle seguenti classi:

- `ListenerFinestraPrincipale` (package `gui`): alla pressione del pulsante deve corrispondere l'avvio dell'attività principale.
- `Gara` (package `ciclisti`)
- `CiclistaFired` (package `ciclisti.ciclista`)
- `AttivitaPrincipale` (package `attivita_composte`)
- `IscriviCiclista` (package `attivita_atomiche`)
- `ManagerPartecipa` (package `ciclisti.partecipa`)

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 *Ciclista*

```

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

InizioSpecificaTransizioniClasse Ciclista
  Transizione: pronto -> inGara
    start/corri{dest:this}
  Evento: start
  Condizione: --
  Azione:
    pre: evento.dest = this and  $\exists l \in Partecipa \mid l.ciclista = this$ 
    post: nuovoevento = corri{mitt=this, dest=this} and l.kmPercorsi = 0

  Transizione: inGara -> inGara
    corri{!primo^!arrivato}/corri{dest:this}
  Evento: corri
  Condizione: ([!primo^!arrivato])

```

Sia $l \in Partecipa \mid l.ciclista = \text{this}$.
 $\forall l' \in Partecipa \mid (l'.gara = l.gara) \rightarrow l'.kmPercorsi < l'.gara.distanza \wedge \text{-- !arrivato}$
 $\exists l' \in Partecipa \mid l'.ciclista \neq \text{this} \wedge l'.gara = l.gara \wedge l'.kmPercorsi < l'.kmPercorsi \text{ -- !primo}$.

Azione:

pre: evento.dest = this and $\exists l \in Partecipa \mid l.ciclista = \text{this}$
post: nuovevento = corri{mitt=this, dest=this} and $l \notin Partecipa$ and
 $\exists l' \in Partecipa \mid l'.ciclista = l.ciclista \wedge l'.gara = l.gara \wedge$
 $l'.kmPercorsi = l.kmPercorsi + (l.gara.distanza/100) \cdot (1 + \text{rand}([0, 1]))$

Transizione: inTesta -> inTesta

corri[primo^!arrivato]/corri{dest:this}

Evento: corri

Condizione: ([primo^!arrivato])

Sia $l \in Partecipa \mid l.ciclista = \text{this}$.
 $\forall l' \in Partecipa \mid (l'.gara = l.gara) \rightarrow l'.kmPercorsi < l'.gara.distanza \wedge \text{-- !arrivato}$
 $\forall l' \in Partecipa \mid (l.ciclista \neq \text{this} \wedge l.gara = l'.gara) \rightarrow l'.kmPercorsi \leq l.kmPercorsi \text{ -- primo}$

Azione:

pre: evento.dest = this and $\exists l \in Partecipa \mid l.ciclista = \text{this}$
post: nuovevento = corri{mitt=this, dest=this} and $l \notin Partecipa$ and
 $\exists l' \in Partecipa \mid l'.ciclista = l.ciclista \wedge l'.gara = l.gara \wedge$
 $l'.kmPercorsi = l.kmPercorsi + (l.gara.distanza/100) \cdot (1.1 - \text{rand}([0, 1]))$

Transizione: inGara -> inTesta

corri[primo^!arrivato]/corri{dest:this}

Evento: corri

Condizione: ([primo^!arrivato])

-- Analoga alla condizione di inTesta -> inTesta

Azione:

-- Analoga all'azione di inGara -> inGara

Transizione: inTesta -> inGara

corri[!primo^!arrivato]/corri{dest:this}

Evento: corri

Condizione: ([!primo^!arrivato])

-- Analoga alla condizione di inGara -> inGara

Azione:

-- Analoga all'azione di inTesta -> inTesta

Transizione: inGara -> finito, inTesta -> finito

corri[arrivato]

Evento: corri

Condizione: ([arrivato])

Sia $l \in Partecipa \mid l.ciclista = \text{this}$.
 $\exists l' \in Partecipa \mid (l'.gara = l.gara) \wedge l'.kmPercorsi \leq l.gara.distanza \text{ -- arrivato}$

Azione:

pre: evento.dest = this and $\exists l \in Partecipa \mid l.ciclista = \text{this}$
post: --

FineSpecifica

Attività di I/O

InizioSpecificaAttivitàAtomica InserisciDatiGara

InserisciDatiGara ():(Gara)

pre: --

post: Legge nome e distanza partenza-traguardo di una gara, forniti in input dall'utente.
result è la gara creata a partire dai dati inseriti.

FineSpecifica

InizioSpecificaAttivitàAtomica InserisciDatiCiclista

InserisciDatiCiclista ():(Ciclista)

```
pre: --
post: Legge il nome del ciclista, fornito in input dall'utente.
      result è il ciclista creato a partire dai dati inseriti.
FineSpecificica
```

```
InizioSpecificicaAttivitàAtomica ChiediSeAltroCiclista
ChiediSeAltroCiclista ():(Bool)
pre: --
post: Chiede all'utente se vuole iscrivere un altro ciclista alla gara.
      result è true in caso affermativo, false altrimenti.
FineSpecificica
```

```
InizioSpecificicaAttivitàAtomica VisualizzaGara
VisualizzaGara (g:Gara):()
pre: --
post: mostra la finestra di visualizzazione della gara
FineSpecificica
```

```
InizioSpecificicaAttivitàAtomica ProclamaVincitori
ProclamaVincitore (g:Gara):()
pre: --
post: Stampa i nomi dei vincitori della gara g
FineSpecificica
```

Attività Atomiche

```
InizioSpecificicaAttivitàAtomica IscrivitiCiclista
IscriviCiclista (c:Ciclista,g:Gara) : ()
pre: --
post:
      -- viene creato un link  $l$  di tipo Partecipa tale che  $l.kmPercorsi = 0, l.gara = g, l.ciclista = c$ 
FineSpecificica
```

```
InizioSpecificicaAttivitàAtomica AvviaGara
AvviaGara(g:Gara):()
pre: --
post:
      -- inizializza l'Environment, inserendovi tutti i ciclisti partecipanti alla gara g (come Listener) a cui invia l'evento start;
      -- successivamente, attiva i Listener.
FineSpecificica
```

```
InizioSpecificicaAttivitàAtomica TestFine
TestFine(g:Gara):(Bool)
pre: --
post:
      -- result è true se tutti i partecipanti alla gara g sono nello stato finito,
      -- e false altrimenti.
FineSpecificica
```

```
InizioSpecificicaAttivitàAtomica AggiornaVincitori
TestFine(g:Gara):()
pre: --
post:
      Per ogni link  $l$  di tipo Partecipa tale che  $l.gara = g$ ,
      se  $l.kmPercorsi = \max\{m.kmPercorsi \mid m \in Partecipa \wedge m.gara = g\}$  allora  $l \in Vincitore$ 
FineSpecificica
```

Attività Composte

InizioSpecificaAttività AttivitaPrincipale

```
AttivitaPrincipale():()
```

Variabili Processo:

```
gara: Gara -- gara corrente
ciclista: Ciclista -- ciclista corrente
altro: Bool -- altro ciclista da aggiungere?
inCorso: Bool -- la gara è ancora in corso?
```

Inizio Processo:

```
InserisciDatiGara():(gara);
InserisciDatiCiclista():(ciclista);
IscriviCiclista(ciclista,gara):();

do{
  InserisciDatiCiclista():(ciclista);
  IscriviCiclista(ciclista,gara):();
  ChiediSeAltroCiclista():(altro);
}while{altro}

VisualizzaGara(gara):();
AvviaGara(gara):();

do{
  -- Attendi qualche (e.g., 100) millisecondo
  TestFine():(inCorso);
}while{inCorso}

AggiornaVincitori(gara):();
ProclamaVincitori(gara):();
```

FineSpecifica

Progetto

Responsabilità sulle Associazioni

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

| Associazione | Classe | Ha Responsabilità |
|--------------|----------|-------------------|
| partecipa | Gara | SÌ (O,M) |
| | Ciclista | SÌ (O,M) |
| vincitore | Gara | SÌ (O,M) |
| | Ciclista | 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 e la tabella delle assunzioni sulla nascita.

| Classe UML | Proprietà Immutabile | Proprietà | | |
|-------------------|-----------------------------|------------------------------|---|---|
| Classe UML | Nota alla nascita | Non nota alla nascita | | |
| Gara | nome distanza | - | - | - |
| Ciclista | 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.