SAPIENZA - Università di Roma
Facoltà di Ingegneria – Corso di Laurea in Ingegneria Informatica
Esercitazioni di Progettazione del Software - A.A. 2008/2009
Prova al calcolatore del 15 gennaio 2010

Requisiti

Si vuole realizzare un'applicazione che simuli la proliferazione di colonie in alcune regioni geografiche. Ciascuna regione è quadrata e caratterizzate dal nome e dalla misura di uno dei suoi lati. Essa contiene un insieme di *colonie*, disposte secondo *una griglia* quadrata di dimensione $lato \times lato$ ($lato \geq 2$). Di esse, interessano le relazioni di adiacenza reciproca lungo *righe* e *colonne*. In Figura 1(a) è riportato il diagramma UML delle classi per l'applicazione considerata. Le associazioni *riga* e *colonna* modellano le relazioni di adiacenza.

Due colonie di una stessa regione sono dette *vicine* se sono adiacenti oppure entrambi adiacenti ad una terza colonia, ma appartenenti a righe e colonne diverse. Ad esempio, tutte le colonie in Figura 1(b) contrassegnate dalla lettera D sono vicine alla colonia C, mentre quelle contrassegnate dalla lettera E non sono vicine a C, in quanto due colonie contrassegnate dalle lettere C ed E giacciono sempre su una stessa riga o colonna. Chiaramente, se C è vicina a D vale anche il viceversa.

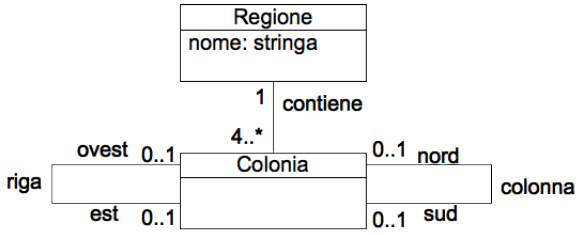
Una colonia può essere in uno stato tra: *deserta*, *popolata* e *instabile*. Inizialmente, ciascuna colonia è nello stato *deserta*. La proliferazione ha luogo come segue. Quando una colonia C è nello stato *deserta*: se il numero n_P delle colonie ad essa adiacenti che sono nello stato *popolata* è $n_P = 0$, allora C va nello stato *instabile*; altrimenti, rimane nello stato *deserta*. Quando C è nello stato *instabile*: se n_P (v. sopra) ed n_I –cioè, il numero delle colonie adiacenti a C che sono nello stato *instabile*– sono tali che $1 \leq n_P + n_I \leq 5$ allora C va nello stato *popolata*; altrimenti, va nello stato *deserta*. Infine, quando C è nello stato *popolata*: se $n_P = 9$ o $n_D > 7$, dove n_D è il numero delle colonie adiacenti a C che sono nello stato *deserta*, allora C va nello stato *deserta*; altrimenti, rimane nello stato *popolata*.

In Figura 1(c) è riportato il diagramma UML degli stati e delle transizioni della classe *Colonia*.

L'applicazione deve:

- creare una nuova regione e le relative colonie (configurando opportunamente le adiacenze reciproche), a partire da dati (nome e dimensione del lato della regione) forniti in input;
- simulare l'evoluzione delle colonie della regione, partendo dalla situazione in cui tutte le colonie sono nello stato *deserta*;
- terminare l'esecuzione della simulazione, quando specificato dall'utente;
- stampare, al termine della simulazione, la *densità abitativa* della regione, ovvero il rapporto, al momento dell'interruzione, tra il numero di celle che si trovano nello stato *popolata* ed il numero totale di celle.

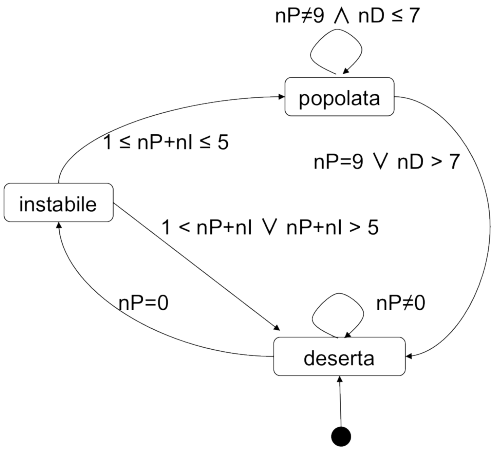
In Figura 1(d) è riportato il diagramma delle attività corrispondente.



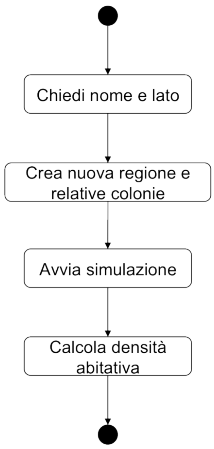
(a) Diagramma UML delle classi

		E		
	D	D	D	
E	D	C	D	E
	D	D	D	
		E		

(b) Esempio di colonie vicine



(c) Diagramma UML degli stati e delle transizioni della classe *Colonia*



(d) Diagramma UML delle attività

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`)
- `TipoLinkColonna` (package `colonie`)
- `ManagerColonna` (package `colonie`)
- `Colonia` (package `colonie`)
- `ColoniaFired` (package `colonie.colonia`)
- `OperazioniUtente` (package `operazioniutente`)

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

Operazioni Utente

```
InizioSpecificaOperazioniClasse OperazioniUtente
  context OperazioniUtente::densitaAbitativa(Regione r): float
    pre:--
    post: popolate=r.contiene.Colonia->select(c|c^getStato.result = popolata)->sum()
          and
          totali = r.contiene.Colonia->sum() and result = popolate/totali
FineSpecifica
```

Segnatura delle Attività di I/O

```
InizioSpecificaOperazioniIO AttivitaPrincipale

  operazione: leggiRegione
    -- Legge nome e lato di una regione, forniti in input dall'utente
    in: --
    out: recordRegione : RecordRegione

  operazione: attendiFineSimulazione
    -- Rimane in attesa che l'utente faccia terminare la simulazione
    in: --
    out: --

FineSpecifica
```

NOTE:
`RecordRegione` è un record con due campi, `nome` (di tipo `String`) e `lato` (di tipo `int`), usati per memorizzare il valore dei campi dato di una regione.

Benché nell'implementazione delle attività di I/O compaiano altri parametri di input, essi non sono riportati nella specifica in quanto usati esclusivamente per ragioni implementative. La loro presenza non è rilevante e può essere trascurata.

Definizione delle Variabili dell'Attività Principale

```
InizioSpecificaVariabiliAttività AttivitaPrincipale
  recordRegione: RecordRegione -- dati della regione forniti in input
  regione: Regione -- la regione
  colonieInizialmentePopolate : Set(Colonia) -- le colonie inizialmente popolate
  densita : float -- densita abitativa della regione al termine della simulazione
FineSpecifica
```

Definizione delle Attività Atomiche

```
InizioSpecificaAttività CreaRegione
    -- Crea una nuova regione a partire dai dati inseriti in input e memorizzati in recordRegione,
    -- crea le colonie corrispondenti e ne imposta le adiacenze secondo una disposizione a griglia.
    pre: --
    post: Regione.contains(nuovaRegione) and nuovaRegione.nome=recordRegione.nome and
```

```
      nuovaRegione.lato=recordRegione.lato and
      Regione.allInstances() = Regione@pre.allInstances() -> including(nuovaRegione) and
      -- nuovaRegione contiene un numero di nuove colonie pari a recordRegione^2,
      -- disposte a griglia
FineSpecifica

InizioSpecificaAttività CalcolaDensitaAbitativa
    -- Esegue l'operazone utente densitaAbitativa e
    -- ne memorizza il risultato in densita
    pre: --
    post: densita=^OperazioniUtente.densitaAbitativa.result
FineSpecifica
```

NOTE:
Altre eventuali attività presenti nell'implementazione possono essere trascurate.

Progetto

Responsabilità sulle Associazioni

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

Associazione	Classe	Ha Responsabilità
contiene	Regione	SÌ (O,M)
	Colonia	SÌ (M)
colonna	nord	SÌ (O,M)
	sud	SÌ (O,M)
riga	est	SÌ (O,M)
	ovest	SÌ (O,M)

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
Regione	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.