

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 26 giugno 2009

## Requisiti

Si vuole realizzare un'applicazione che simuli la proliferazione di colonie in alcune regioni geografiche. Ciascuna regione è quadrata e caratterizzata 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.

La proliferazione delle colonie ha luogo come segue. Inizialmente, ciascuna colonia è nello stato *nonAttiva*. Da questo, può ricevere gli eventi *popola* o *svuota*. Il primo porta la colonia nello stato *popolata* mentre il secondo nello stato *deserta*. Nello stato *popolata*, se 2 o 3 delle sue colonie vicine sono nello stato *popolata*, allora rimane nello stato *popolata*, altrimenti passa nello stato *deserta*. Quando è nello stato *deserta*, se esattamente 3 delle sue colonie vicine sono nello stato *popolata* allora cambia il proprio stato in *popolata*, altrimenti rimane nello stato *deserta*. Non vi sono vincoli sull'ordine con cui le colonie cambiano stato.

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

L'applicazione deve:

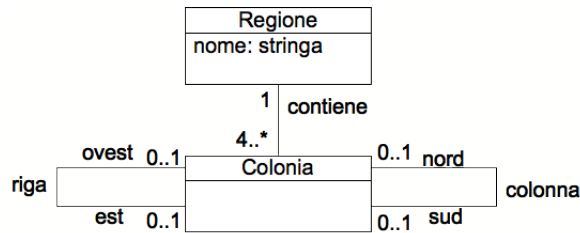
- creare una nuova regione e le relative colonie (configurando opportunamente le adiacenze reciproche), a partire da dati (nome e dimensione del lato della regione) forniti in input;
- impostare le colonie inizialmente popolate, secondo dati forniti in input (tramite interfaccia grafica);
- simulare l'evoluzione delle colonie della regione, partendo dalla situazione specificata al punto precedente;
- terminare l'esecuzione della simulazione, quando specificato dall'utente;
- stampare, al termine della simulazione, il numero di colonie popolate al momento dell'interruzione.

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

In ogni momento deve essere possibile creare una nuova regione e le relative colonie, ed avviare una nuova simulazione, anche contemporaneamente alla creazione di altre regioni o mentre altre simulazioni sono in corso.

---

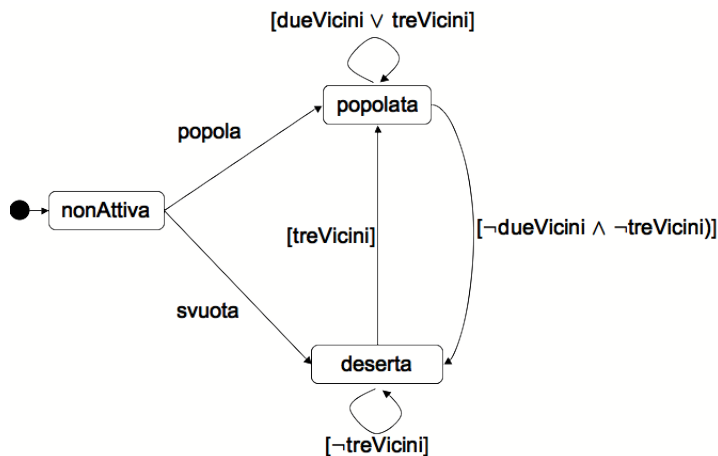
**La prova consiste nel completare o modificare il codice fornito insieme al testo, in modo da soddisfare i requisiti sopra riportati. Seguendo le indicazioni riportate nei commenti al codice, si chiede di intervenire sulle seguenti classi:**



(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

- ListenerFinestraPrincipale (package gui)
- TipoLinkContiene (package colonie)
- ManagerContiene (package colonie)
- Regione (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::quanteColoniePopolate(Regione r): int
pre:--
  
```

```
post: result=r.contiene.Colonia->select(c|c^getStato and c^getStato.result = popolata)->sum()
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: impostazioneIniziale
-- Legge quali colonie sono inizialmente popolate, secondo l'input fornito dall'utente
in: --
out: colonieInizialmentePopolate : Set(Colonia)

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
coloniePopolate : int -- numero di colonie popolate 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 recordRegione2,
      -- disposte a griglia
```

## FineSpecifica

### InizioSpecificaAttività ContaColoniePopolate

```
-- Esegue l'operazione utente quanteColoniePopolate e
-- ne memorizza il risultato in coloniePopolate
pre: --
post: ^OperazioniUtente.quanteColoniePopolate and
      coloniePopolate=^OperazioniUtente.quanteColoniePopolate.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	Proprietà	
		Nota alla nascita	Non nota alla nascita
Regione	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.