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

Corso di Progettazione del Software

Esame del **13 settembre 2013**

Tempo a disposizione: 3 ore

Requisiti. L'applicazione da progettare riguarda la gestione di un club cibernetico di cuori solitari. Un cliente ha un nome e una homepage (una stringa che denota la url). I clienti sono partizionati secondo il genere in uomini e donne. Ad un cliente piacciono, con un gradazione denotata da un intero tra 1 e 10, altri clienti che però devono necessariamente essere del genere opposto. Si noti che se al cliente A piace la cliente B questo non vuol dire che necessariamente alla cliente B piaccia il cliente A. Una serata di incontro è caratterizzata da un nome, un tema (una stringa) e una data. I clienti partecipano a 0 o più serate.

Il cliente è inizialmente in uno stato di *riposo*. Quando è in *riposo* può riceve un evento *inizio-serata* (con parametro la serata) e in tal caso verifica di partecipare effettivamente alla serata, e in caso affermativo passa nello stato *disponibile*. Nello stato *disponibile* può ricevere l'evento *scelto* o l'evento *muoviti*. Se riceve l'evento *scelto* e il mittente è un cliente (che partecipa alla serata) dell'altro genere che gli piace allora lancia l'evento *accetto* e passa nello stato *riposo*, altrimenti lancia l'evento *rifiuto* e rimane nello stato *disponibile*. Se, nello stato *disponibile*, riceve l'evento *muoviti*, lancia un evento *scelto* ad un cliente dell'altro genere scelto tra quelli che gli piacciono e partecipano alla serata (secondo un algoritmo proprietario che si assume dato) e passa nello stato *attesa di risposta*. In tale stato se riceve l'evento *accetto* dal cliente selezionato passa nello stato *riposo*, mentre se riceve l'evento *rifiuto* (sempre dal cliente selezionato) passa nello stato *disponibile*. In ogni stato diverso da *riposo* il cliente può ricevere l'evento *fine-serata* e in tal caso passa nello stato di *riposo*.

Siamo interessati alla seguente attività principale che prende come parametro una serata. L'attività inizia attivando concorrentemente le seguenti due sottoattività: *(i)* esecuzione e *(ii)* analisi. La sottoattività esecuzione *(i)* manda in esecuzione tutti i clienti che partecipano alla serata inviando a tutti un evento *inizio-serata* (con parametro la serata stessa) e in aggiunta ad alcuni (scelti con un metodo proprietario i cui dettali non interessano) l'evento *muoviti*. Poi si mette in attesa del comando di chiusura della serata da parte dell'utente che interrompe l'esecuzione mandando l'evento *fine-serata* a tutti i clienti. La sottoattività di analisi *(ii)* calcola le coppie di clienti che partecipano alla serata che si piacciono reciprocamente e le stampa. Una volta che tali sottoattività sono state completate stampa un messaggio di saluto per chiudere la serata.

Domanda 1. Basandosi sui requisiti riportati sopra, effettuare la fase di analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe *Cliente*, diagramma delle attività, specifica del diagramma stati e transizioni, e specifica della attività principale e delle sottoattività NON atomiche (indicando in modo esplicito quali attività atomiche sono di I/O e quali sono Task), motivando, qualora ce ne fosse bisogno, le scelte effettuate.

Domanda 2. Effettuare la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate. È obbligatorio definire solo le responsabilità sulle associazioni del diagramma delle classi.

Domanda 3. Effettuare la fase di realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte effettuate. È obbligatorio realizzare in JAVA solo i seguenti aspetti dello schema concettuale:

- La classe `Cliente`, la sottoclasse `Donna`, la classe `ClienteFired`, e le classi per rappresentare le *associazioni* che le legano con le altre classi.
- L'*attività principale* e le sue eventuali sottoattività NON atomiche.