Sapienza - Università di Roma
Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Esercitazioni di Progettazione del Software - A.A. 2009/2010

Prova al calcolatore del 4 Febbraio 2011

Requisiti

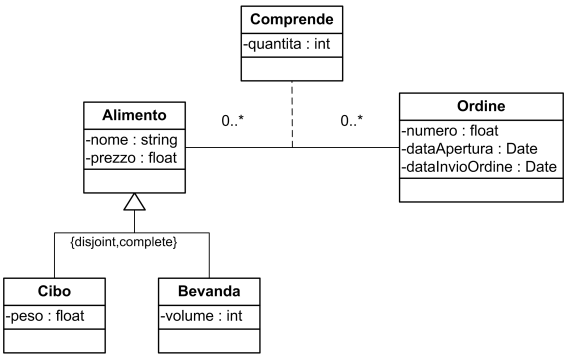

Figura 1: Diagramma UML delle classi

Si vuole realizzare un'applicazione per una società di vendita di alimenti per la gestione degli ordini. Ciascun alimento è caratterizzato dal nome e dal prezzo di acquisto. Esistono due tipi di alimenti: cibi e bevande. In aggiunta al nome e al prezzo, dei primi interessa il peso mentre delle seconde il volume (in cc). Ogni ordine è caratterizzato da un numero, dalla data di apertura e dalla data in cui l'ordine verrà evaso per essere inviato al cliente. Il diagramma delle classi corrispondente è in Figura 1.

L'interazione dell'utente dell'applicazione ha luogo tramite una interfaccia grafica. Prima dell'apertura dell'ordine stesso, viene chiesto all'utente se questi è interessato ad aggiungere all'ordine cibi e/o bevande. Se l'utente decide di non essere interessato a nessuno dei due, l'ordine viene considerato annullato e la sessione di interazione termina, senza creare l'ordine. Se l'utente dimostra interesse in uno tra cibi e bevande, il processo continua con l'esecuzione del task *Creare Ordine*. Successivamente, il processo è composto da due rami eseguiti in parallelo che si occupano dell'aggiunta di bevande o cibi all'ordine. I due rami sono simili: se l'utente ha dimostrato disinteresse in aggiungere cibi o bevande, allora il corrispondente ramo finisce senza eseguire alcun task. Per il ramo cibi o bevande, se l'utente ha mostra interesse, viene eseguita un'attività di I/O per chiedere il cibo/bevanda di interesse; una volta scelto il cibo/bevanda, questo viene aggiunto all'ordine tramite il task *Creare Ordine*. Successivamente viene chiesto con un'ulteriore attività di I/O se si desidera scegliere un ulteriore cibo/bevanda. Se si risponde negativamente, il ramo si conclude.Se si risponde positivamente, viene chiesto nuovamente il cibo/bevanda di interesse, il quale viene aggiunto con un task, e successivamente viene posta nuovamente la domanda se si desidera scegliere un ulteriore cibo/bevanda.

Quando entrambi i rami si concludono, viene calcolata la quantità totale di alimenti dell'ordine e l'importo complessivo dello stesso. Successivamente viene eseguita una attività di I/O che visualizza i dettagli dell'ordine e l'importo totale. A questo punto l'ordine può essere confermato o meno. Se l'ordine non viene confermato, questo **non** viene cancellato ma viene chiesto ancora all'utente di modificarlo. Se ordine è confermato, lo stesso viene preparato per l'invio; la preparazione per l'invio comporta anche il calcolo della data quando lo stesso verrà evaso. Il ritardo tra la chiusura può dipendere dalla assenza di prodotti dell'ordine in magazzino e, quindi, dalla necessità del loro successivo riordine presso i fornitori.

Il processo termina mostrando all'utente nuovamente l'ordine, il quale contiene a questo punto anche la data di evasione.

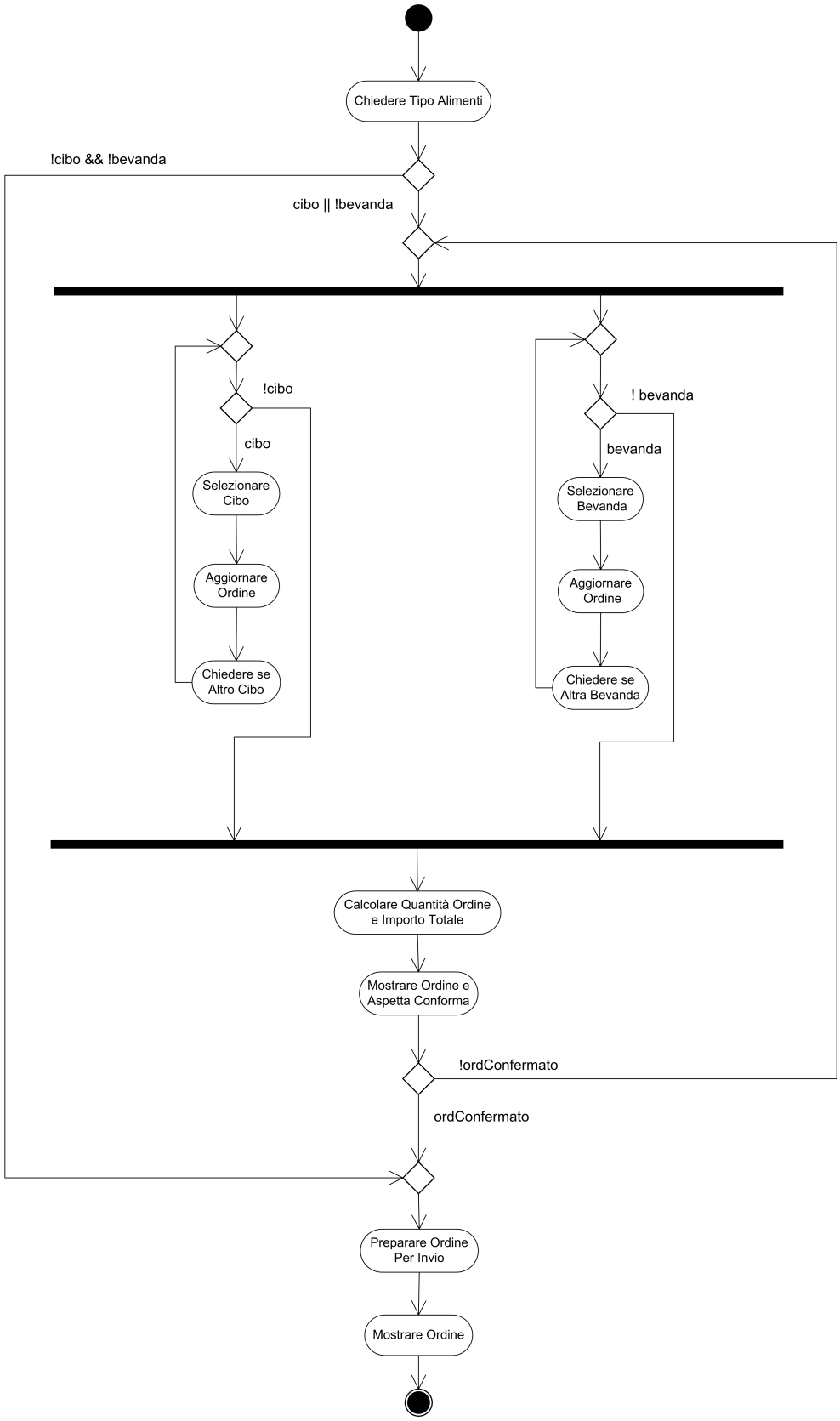

Figura 2: Diagramma UML delle attività

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:

- `Alimento` (package `venditore`)
- `ManagerComprende` (package `venditore`)
- `AttivitaPrincipale` (package `processo`)
- `AttivitaSottoramoBevande` (package `processo`)
- `CalcolaQuantitaETotale` (package `processo`)
- `AggiornareOrdine` (package `processo`)
- `OperazioniUtente` (package `venditore`)

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

`InizioSpecificaOperazioniUtente OperazioniUtente`

`GetQuantitaVendute (s: Alimento):(int)`

`pre: s!=null`

`post:` Restituisce la quantità totale dell'alimento *s* che è stata venduta in tutti gli ordini.

`FineSpecifica`

Attività di I/O

`InizioSpecificaAttivitàAtomica ChiedereTipoAlimenti`

`InserisciDatiGara ():(RecordTipoAcquisti)`

`pre: --`

`post:` Apre una finestra di dialogo in cui l'utente può specificare quale tipi di alimenti possono essere aggiunti all'ordine.
Restituisce un oggetto di tipo RecordTipoAcquisti che contiene due metodi:
isCibo() se l'utente ha specificato i cibo come possibile alimento
isBevande() se l'utente ha specificato le bevande come possibile alimento
(è possibile che allo stesso `isCibo()=true` e `isBevanda()=true`)

`FineSpecifica`

`InizioSpecificaAttivitàAtomica SelezionareCibo`

`InserisciDatiCiclista ():(RecordAlimento)`

`pre: --`

`post:` Apre una finestra di dialogo in cui l'utente può specificare quale cibo aggiungere l'ordine e la quantità da aggiungere.
Restituisce un oggetto di tipo RecordAlimento che contiene due metodi:
getAlimento() che restituisce l'oggetto di tipo *Alimento* (di fatto, della sottoclasse *Cibo*) corrispondente al cibo selezionato
getQuantita() che restituisce la quantità selezionata

`FineSpecifica`

InizioSpecificaAttivitàAtomica SelezionareBevanda
    InserisciDatiCiclista ():(RecordAlimento)
    pre: --
    post: Apre una finestra di dialogo in cui l'utente può specificare quale cibo aggiungere l'ordine e la quantità da aggiungere.
          Restituisce un oggetto di tipo RecordAlimento che contiene due metodi:+
          getAlimento() che restituisce l'oggetto di tipo *Alimento* (di fatto, della sottoclasse *Bevanda*) corrispondente
          alla bevanda selezionata
          getQuantita() che restituisce la quantità selezionata
FineSpecifica

InizioSpecificaAttivitàAtomica ChiedereSeAltroCibo
    InserisciDatiCiclista ():(Boolean)
    pre: --
    post: Apre una finestra di dialogo in cui viene chiesto all'utente se desidera aggiungere altro cibo all'ordine.
          Restituisce true se l'utente ha scelto di aggiungere altro cibo, false altrimenti.
FineSpecifica

InizioSpecificaAttivitàAtomica ChiedereSeAltraBevanda
    ChiedereSeAltraBevanda ():(Boolean)
    pre: --
    post: Apre una finestra di dialogo in cui viene chiesto all'utente se desidera aggiungere altro cibo all'ordine.
          Restituisce true se l'utente ha scelto di aggiungere altro cibo, false altrimenti.
FineSpecifica

InizioSpecificaAttivitàAtomica ConfermareOrdine
    ConfermareOrdine (int,Date,float, HashSet<RecordAlimento>):(Boolean)
    pre: -- I parametri del metodo sono nell'ordine:
          1) Il numero dell'ordine
          2) La date di apertura dell'ordine
          3) L'importo totale dell'ordine
          4) Un insieme di oggetti *RecordAlimento*, composto da una coppia (Alimento, quantità).
             Per maggiori dettagli su la classe RecordAlimento
          vedasi SelezionareBevanda:

    post: Apre una finestra di dialogo in cui viene mostrato l'ordine e chiesto se l'ordine deve essere confermato come è stato creato
          oppure altri prodotti devono essere ancora aggiunti
          Restituisce true se l'utente ha scelto di confermare l'ordine così come è, false se si desidera aggiungere altri prodotti.
FineSpecifica

InizioSpecificaAttivitàAtomica MostrareOrdine
    MostrareOrdine (int,Date,Date,float, HashSet<RecordAlimento>):()
    pre: -- I parametri dell'ordine sono:
          1) Il numero dell'ordine
          2) La date di apertura dell'ordine
          3) La date di evasione dell'ordine
          4) L'importo dell'ordine
          5) Un insieme di oggetti RecordAlimento, composto da una coppia (Alimento, quantità). Per maggiori dettagli sulla classe RecordAlimento, vedasi l'attività di I/O SelezionareBevanda.
    post: Apre una finestra di dialogo in cui viene mostrato l'ordine, incluso la data di evasione/invio dell'ordine
FineSpecifica

Attività Atomiche

InizioSpecificaAttivitàAtomica CreareOrdine
    CreareOrdine() : (o : Ordine)
    pre: --
    post:
        -- viene creato un oggetto *o* di tipo *Ordine* a cui viene assegnato un numero (progressivo)
FineSpecifica

InizioSpecificaAttivitàAtomica AggiornareOrdine
    AggiornareOrdine(a:Alimento, q:int, o:Ordine):()
    pre: a!=null, q>0, o!=null
    post:
      -- l'alimento o viene aggiunto all'ordine o in quantità q.
FineSpecifica

InizioSpecificaAttivitàAtomica CalcolareQuantitaETotale
    CalcolareQuantitaETotale(o:Ordine):(t:float, a:HashSet<RecordAlimento>) [1]
    pre: --
    post:
      -- t è l'importo totale dell'ordine, calcolato come la somma del prodotto dei prezzi unitari per le rispettive quantità aggiunte
      -- all'ordine, a è un HashSet composto da oggetti *RecordAlimento*, ognuno contenente l'alimento con la rispettiva quantità
FineSpecifica

InizioSpecificaAttivitàAtomica PrepararePerInvio
    TestFine(o:Ordine):()
    pre: o!=null
    post:
      Aggiorna l'ordine o con la data prevista di evasione.[2]
FineSpecifica

Attività Composte

InizioSpecificaAttività AttivitaPrincipale
    AttivitaPrincipale():()
    Variabili Processo:
        corrente: Ordine
        tipoAcquisti: RecordTipoAcquisti
        ancoraCibo: boolean
        ancoraBevande: boolean
        ciboSelezionato: RecordAlimento
        bevandaSelezionata: RecordAlimento
        alimenti: Set<RecordAlimento>
    Inizio Processo:

```
        ancoraBevande=true;
        ancoraCibo=true;
        tipoAcquisti=ChiedereTipoAlimento();
        if(!tipoAcquisti.isBevanda() && !tipoAcquisti.isCibo()) return;
        CreaOrdine():(corrente)
        do {
            fork {
                  t1: while(ancoraBevande) {
                        bevandaSelezionata=SelezionareBevanda();
                        AggiornareOrdine(bevandaSelezionata.getAlimento(),bevandaSelezionata.getQuantita(),ordine):();
                        ancoraBevanda=ChiedereSeAltraBevanda();
                      }
                  t2: while(ancoraCibo) {
                        ciboSelezionato=SelezionareCibo();
                        AggiornareOrdine(ciboSelezionato.getAlimento(),ciboSelezionato.getQuantita(),ordine):();
                        ancoraCibo=ChiedereSeAltroCibo();
                      }
            }
            t1.join();
            t2.join();
            CalcolaQuantitaETotale(corrente):(totale,alimenti);
            ordConfermato=ConfermareOrdine(corrente.getNumero(),corrente.getDataApertura(),totale, alimenti);
        } while(!ordConfermato)
        PrepararePerInvio(corrente):();
        mostrareOrdine(corrente.getNumero(),corrente.getDataApertura(),
             corrente.getDataInvioOrdine(), totale, alimenti);
```

[1] La possibilità per una attività di restituisce più output è ottenuta attraverso due metodi getter, rispettivamente: `getTotale()` e `getAlimenti()`

[2] Per la prova al calcolatore tale data è calcolata in maniera casuale successiva alla data di apertura ordine

Progetto

Responsabilità sulle Associazioni

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

Associazione	Classe	Ha Responsabilità
comprende	Alimento	SÌ (O,M)
	Ordine	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
Ordine	numero
	dataApertura
Alimento	nome
	prezzo
Cibo	peso
Bevanda	volume

	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.