SAPIENZA Università di Roma

A.A. 2008-2009

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Esercitazioni di Progettazione del Software
(Canale A-L)

Diagrammi delle Attività (3)

Fabio Patrizi

Requisiti

Si vuole realizzare un'applicazione per distributori automatici di alimenti. Ciascun alimento è caratterizzato dal nome, dal prezzo e dalla data di scadenza. Esistono solo due tipi di alimenti: cibi e bevande. Dei primi interessa il peso (in grammi) mentre delle seconde il volume (in cc). Durante ciascun acquisto (vedi sotto), l'applicazione mantiene uno scontrino aggiornato. Ogni scontrino comprende gli alimenti acquistati con le relative quantità, ed è caratterizzato da un numero progressivo, dalla data di emissione e dal totale della spesa effettuata, calcolata a partire dagli alimenti che comprende e dalle rispettive quantità. Inoltre, l'applicazione deve mettere a disposizione un'operazione che restituisca la quantità venduta, fino al momento di esecuzione dell'operazione stessa, di un alimento fornito in input.

L'interazione ha luogo tramite interfaccia grafica. All'inizio della sessione d'acquisto, l'applicazione chiede all'utente se ha intenzione di acquistare cibi o bevande. Se l'utente decide di non comprare nulla la sessione termina. Viceversa, se l'utente desidera acquistare cibi o bevande (eventualmente entrambi), viene creato un nuovo scontrino vuoto, con opportuno numero (progressivo) e data di emissione, e l'applicazione mostra le interfacce per l'acquisto degli alimenti indicati. Nel caso si vogliano acquistare sia cibi che bevande, vengono mostrate due interfacce, una per i cibi ed una per le bevande. Esse permettono di specificare un alimento e la quantità che se ne vuole acquistare. L'interazione può aver luogo contemporaneamente sia con l'una che con l'altra interfaccia.

Ogni volta che l'utente seleziona un nuovo alimento per l'acquisto, lo scontrino viene aggiornato con l'alimento scelto e la relativa quantità. Dopodiché, l'applicazione chiede all'utente se ha intenzione di comprare un altro alimento dello stesso tipo. In caso affermativo, ha luogo una nuova iterazione (durante la quale l'utente può acquistare un altro alimento dello stesso tipo appena acquistato). In caso contrario, non sarà più possibile acquistare nuovi alimenti dello stesso tipo, durante la sessione d'acquisto corrente.

Quando l'utente non ha più intenzione di acquistare cibi né bevande, l'applicazione mostra i dettagli dello scontrino con la spesa totale, e la sessione d'acquisto termina.

Si richiede di completare il codice delle classi: `AttivitaSottoramoBevanda`, `AttivitaSottoramoCibo`, `Scontrino`, `OperazioniUtente`, `TipoLinkComprende`, `ManagerComprende`, `LeggiScontrino`, `FinestraPrincipaleListener`, in maniera tale da soddisfare i requisiti sopra descritti.

Diagramma UML delle Classi

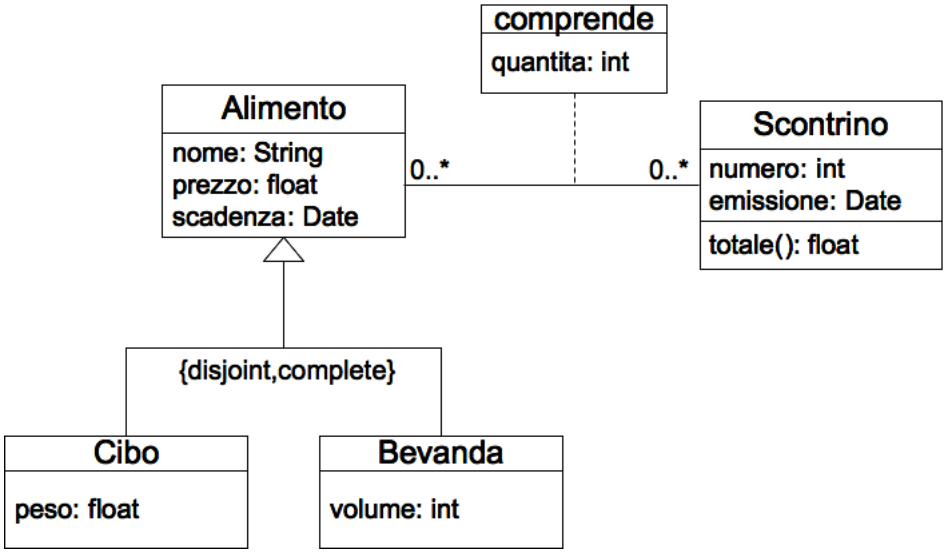
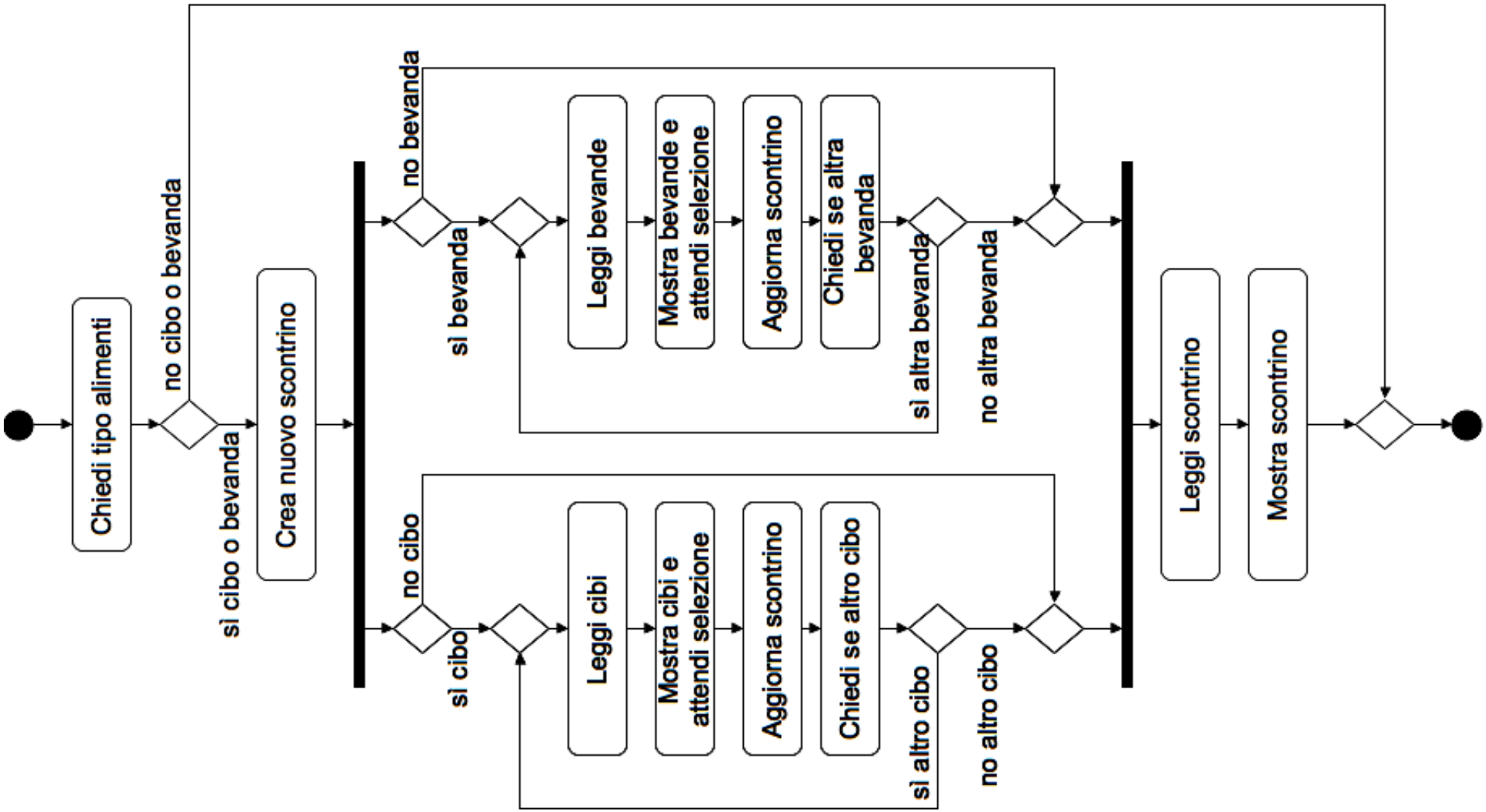


Diagramma delle Attività



Specifica OCL Operazioni della Classe Scontrino

```
InizioSpecificaOperazioniClasse Scontrino
  context Scontrino::totale(): float
    pre: --
    post: sia C l'insieme dei link di tipo comprende in cui self e' coinvolto
    result e' la sommatoria dei termini (c.quantita * c.Alimento.prezzo), calcolata sugli elementi c di  C
```

Specifica OCL Operazioni Utente

```
InizioSpecificaOperazioniClasse OperazioniUtente
  context OperazioniUtente::quantoVenduto(alimento: Alimento): int
    in: Alimento
    out: result = alimento.comprende.quantita -> sum()
```

Specifica dell'Attività Principale

Composto dalle seguenti definizioni, fornite nel seguito:

- Definizione della segnatura delle operazioni di input/output
- Definizione delle variabili dell'attività
- Specifica delle sottoattività

Definizione delle Operazioni Input/Output

```
InizioSpecificaOperazioniIO AttivitaPrincipale
  operazione: chiediTipoAlimenti
  -- da' all'utente la possibilita' di selezionare le voci "cibi" e "bevande"
    in: --
    out: tipoAcquisti: RecordTipoAcquisti --v. sotto

  operazione: selezioneCibo
  -- mostra la lista dei cibi, e permette di selezionarne uno indicandone la quantita' desiderata
    in: listaCibi: Set<String>
    out: alimentoSelezionato: RecordAlimento -- v. sotto

  operazione: selezioneBevande
  -- mostra la lista delle bevande, e permette di selezionarne una indicandone la quantita' desiderata
    in: listaBevande: Set<String>
    out: alimentoSelezionato: RecordAlimento -- v. sotto

  operazione: altroCibo
  -- verifica se l'utente desidera acquistare altro cibo
    in: --
    out: ancoraCibo: boolean
```

(segue)

Definizione delle Operazioni Input/Output (2)

```
  operazione: altraBevanda
  -- verifica se l'utente desidera acquistare un'altra bevanda
    in: --
    out: ancoraBevanda: boolean

  operazione: mostraScontrino
  -- mostra all'utente i dettagli dello scontrino d'acquisto
    in: datiScontrino: RecordScontrino --v. sotto
    out: --
FineSpecifica
```

`RecordTipoAcquisti` è un record con due campi booleani: `cibi` e `bevande`, `true` sse l'utente ha dichiarato l'intenzione di voler acquistare il tipo di elemento cui fanno riferimento

`RecordAlimento` è un record con due campi: una stringa `nome`, contenente il nome dell'alimento acquistato, ed un intero `quantita` contenente la quantità dell'alimento acquistato

`RecordScontrino` è un record con i campi: `numero` (`int`), `emissione` (`Date`), `totale` (`float`) e comprende (`Set<RecordAlimento>`), in cui sono memorizzate le proprietà di uno scontrino, che sono d'interesse per l'applicazione.

Definizione delle Variabili dell'Attività Principale

```
InizioSpecificaVariabiliAttività AttivitaPrincipale
 scontrinoCorrente: Scontrino
 tipoAcquisti: RecordTipoAcquisti
 ancoraBevanda: boolean
 ancoraCibo: boolean
 alimentoSelezionato: RecordAlimento
FineSpecifica
```

Specifica delle Attività Atomiche

```
InizioSpecificaAttività creaScontrino
  pre: --
  post:
    -- Crea un nuovo oggetto di tipo scontrino, con prossimo numero progressivo e data corrente
    -- Assumiamo date le funzioni dataCorrente e prossimoNumero, con ovvia semantica
    nuovoScontrino.oclIsNew() and ^dataCorrente and nuovoScontrino.emissione=dataCorrente.out.data and
    ^prossimoNumero and nuovoScontrino.numero=prossimoNumero.out.numero and
    Scontrino.allInstances() = Scontrino@pre.allInstances() -> including(nuovoScontrino) and
    scontrinoCorrente=nuovoScontrino
FineSpecifica
```

```
InizioSpecificaAttività leggiCibi
  --Crea un insieme di stringhe contenente i nomi di tutti i cibi presenti
  pre: --
  post: risultato=Cibo.allInstances().nome
FineSpecifica
```

```
InizioSpecificaAttività leggiBevande
  --Crea un insieme di stringhe contenente i nomi di tutte le bevande presenti
  pre: --
  post: risultato=Bevanda.allInstances().nome
FineSpecifica
```

(segue)

Specifica delle Attività Atomiche (2)

```
InizioSpecificaAttività aggiornaScontrino
  --Aggiorna scontrinoCorrente a partire dal contenuto di alimentoSelezionato
  pre: --
  post: nuovoLinkComprende.oclIsNew() and comprende.allInstances()->include(nuovoLinkComprende) and
        nuovoLinkComprende.Scontrino=scontrinoCorrente and
        nuovoLinkComprende.quantita=alimentoSelezionato.quantita and
        nuovoLinkComprende.Alimento=Alimento.allInstances()->select(nome=alimentoSelezionato.risultato.nome) a
        comprende=comprende@pre.allInstances()->including(nuovoLinkComprende)
FineSpecifica
```

```
InizioSpecificaAttività leggiScontrino
  pre: --
  post: -- Crea un RecordScontrino a partire dalle proprieta' dello scontrinoCorrente
FineSpecifica
```

Fase di progetto

Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

- 1. Requisiti
- 2. Specifica delle operazioni
- 3. Vincoli di molteplicità nel diagramma delle classi.

Associazione	Classe	ha resp.
comprende	*Scontrino*	SÌ[2]
	Alimento	SÌ[2]

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, a causa:

- dei vincoli di molteplicità 0..* delle associazioni,
- delle variabili necessarie per vari algoritmi.

Per fare ciò, utilizzeremo le classi del collection framework di Java: `Set`, `HashSet`.

Corrispondenza fra tipi UML e Java

Riassumiamo le nostre scelte nella seguente tabella di corrispondenza dei tipi UML.

Tipo UML	Rappresentazione in Java
String	`String`
booleano	`boolean`
int	`int`
float	`float`
Date	`Date`
Insieme	`HashSet`

Tabelle di gestione delle proprietà di classi UML

Riassumiamo le nostre scelte differenti da quelle di default mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita*.

Classe UML	**Proprietà immutabile**
Alimento	*nome, prezzo, scadenza*
Cibo	*peso*
Bevanda	*volume*
Scontrino	*numero, emissione*

	Proprietà	
Classe UML	**nota alla nascita**	**non nota alla nascita**
-	-	-

Altre considerazioni

Sequenza di nascita degli oggetti: Non dobbiamo assumere una particolare sequenza di nascita degli oggetti.

Valori alla nascita: Non sembra ragionevole assumere che per qualche proprietà esistano valori di default validi per tutti gli oggetti.

API delle classi Java progettate

Derivano immediatamente dalla fase di analisi e dalle precedenti osservazioni (circa responsabilità e gestione delle proprietà).