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

Esercitazioni di Progettazione del Software
A.A. 2012/2013

Prova al calcolatore del 31 maggio 2013

Requisiti

Si vuole realizzare un'applicazione per simulare il percorso di battelli che effettuano crociere turistiche sui fiumi. Una tratta navigabile che costituisce il percorso di un battello è caratterizzata dal nome del fiume, dalla profondità minima e massima del fondale in metri, e comprende un insieme ordinato di punti di attracco, ciascuno dei quali si trova ad una certa distanza in chilometri dall'inizio della tratta. La lunghezza di una tratta è data dalla distanza dell'ultimo attracco, cui viene aggiunto sempre un chilometro che rappresenta la posizione del porto di arrivo rispetto all'ultimo attracco. Per ogni punto di attracco è di interesse sapere il nome e la lunghezza in metri della banchina di attracco. Durante una simulazione un battello percorre una tratta navigabile. Un battello è identificato da un nome ed è caratterizzato dal numero di posti, dalla lunghezza in metri dello scafo e dalla profondità in metri dello scafo (dalla linea di galleggiamento). Quando un battello percorre una tratta, è di interesse conoscere i metri percorsi dal battello sulla tratta stessa.

In Figura 1 è mostrato il diagramma delle classi corrispondente al dominio.

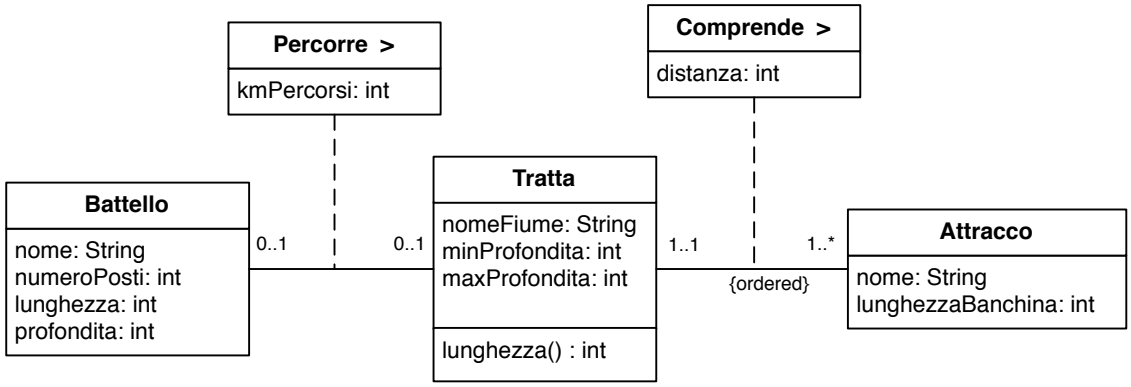

Figura 1: Diagramma UML delle classi

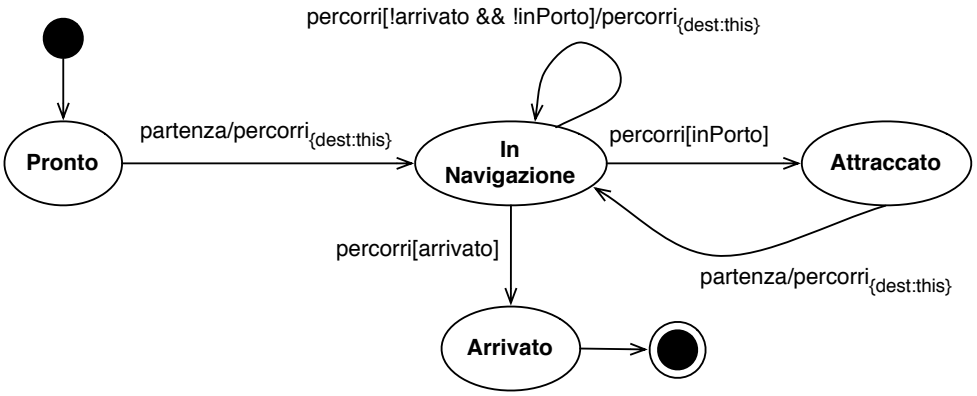

Figura 2: Diagramma degli stati e delle transizioni relativo alla classe *Battello*

Durante una simulazione un battello si comporta come segue. Un battello che percorre una certa tratta si trova inizialmente nello stato *Pronto*, in attesa della partenza (evento *partenza* generato dall'utente); quando riceve l'evento *partenza*, il battello passa nello stato $InNavigazione$ e invia a se stesso l'evento *percorri*, al fine di procedere sulla tratta. Il battello nello stato $InNavigazione$ che riceve l'evento *percorri* si comporta come segue:

- se il battello ha raggiunto un punto di attracco (cioè ha percorso un numero di metri pari alla distanza di un attracco sulla tratta) con una banchina lunga almeno quanto il battello (condizione *inPorto*), il battello si ferma e passa nello stato *Attraccato*;
- se il battello ha completato la sua navigazione (condizione *arrivato*), cioè ha percorso un numero di metri pari alla lunghezza della tratta, il battello si ferma e passa nello stato *Arrivato*;
- se il battello non ha raggiunto un punto di attracco dove può fermarsi né ha completato la sua navigazione, avanza di 100 metri sulla tratta aggiornando i metri percorsi, e manda a se stesso l'evento *percorri*;

Quando il battello si trova nello stato *Attraccato*, resta in attesa dell'evento *Partenza* generato dall'utente; quando riceve tale evento, passa nuovamente nello stato $InNavigazione$, avanza di 100 metri sulla tratta aggiornando i metri percorsi e manda a se stesso l'evento *percorri*.

In Figura 2 è mostrato il diagramma degli stati e delle transizioni relativo alla classe *Battello*.

Una sessione di interazione con l'applicazione si svolge come segue:

- l'utente inserisce i dati della tratta, specificando il nome del fiume e la profondità minima/massima;
- l'utente inserisce i dati del battello, specificandone il nome, il numero di posti, e lunghezza e profondità dello scafo;
- si verifica la navigabilità della tratta rispetto al battello:
  - se la profondità dello scafo è minore della profondità minima della tratta, il battello può percorrere la tratta e si procede come specificato nel seguito;
  - altrimenti, viene mostrato un messaggio di errore e l'applicazione termina;

- il battello viene posizionato sulla tratta da percorrere;
- iterativamente vengono definiti i punti di attracco sulla tratta; in particolare, per ogni punto di attracco da definire sulla tratta:
  - l'utente inserisce i dati del punto di attracco, specificando il nome e la distanza rispetto al punto di attracco precedentemente definito (o rispetto al porto di partenza per il primo punto di attracco);
  - viene aggiunto il punto di attracco sulla tratta;
  - si procede poi con l'eventuale punto di attracco successivo, sulla base della scelta dell'utente;
- dopo aver definito i punti di attracco sulla tratta, viene visualizzata la simulazione tramite opportuna interfaccia grafica che mostra il percorso del battello sulla tratta e consente all'utente di generare gli eventi per controllare la navigazione del battello;
- al termine della simulazione, si procede con l'eventuale simulazione successiva, sulla base della scelta dell'utente.

In Figura 3 è riportato il diagramma delle attività corrispondente.

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[1], si chiede di intervenire sulle seguenti classi:

- `FinestraPrincipale` (package `app.gui`) (si vedano le considerazioni in fondo al documento di specifica)
- `Attracco` (package `app.dominio`)
- `BattelloFired` (package `app.dominio`)
- `VerificaNavigabilita` (package `app.attivita.atomiche`)
- `AttivitaPrincipale` (package `app.attivita.complesse`)

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.

[1] le porzioni di codice su cui intervenire sono identificate dal commento `/* DA COMPLETARE A CURA DELLO STUDENTE */`

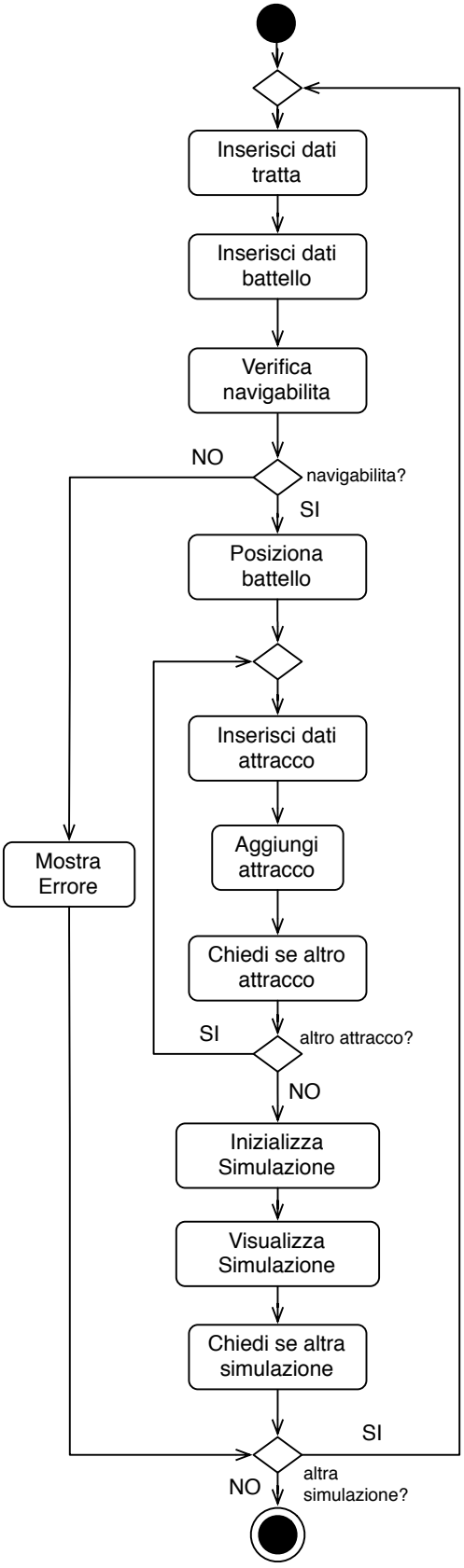

Figura 3: Diagramma delle attività

Analisi

Operazioni Classe `Tratta`

`InizioSpecificaOperazioniClasse` `Tratta`

`InizioSpecificaOperazione` `Lunghezza`
`Lunghezza():(int)`
`pre: --`
`post:` Calcola la lunghezza complessiva della tratta `this`.
Sia C l'insieme ordinato dei link di tipo *Comprende* che coinvolgono la tratta `this`;
`result` vale $lastElementOf(C).distanza + 1000$
`FineSpecifica`

`FineSpecifica`

Specifica del diagramma degli stati e delle transizioni della classe *Battello*

`InizioSpecificaStatiClasse` `Battello`

`Stato`: `{pronto, inNavigazione, attraccato, arrivato}`
`Variabili di stato ausiliarie`: –
`Stato iniziale`:
`stato = pronto`

`FineSpecifica`

`InizioSpecificaTransizioniClasse` `Battello`

`Transizione`: `pronto --> inNavigazione`
`partenza/percorri{dest:this}`
`Evento`: `partenza`
`Condizione`: `--`
`Azione`:
`pre`: `--`
`post`: `nuovoevento = percorri{mitt=this, dest=this}`

`Transizione`: `inNavigazione --> attraccato`
`percorri[inPorto]`
`Evento`: `percorri`
`Condizione`: `([inPorto])`
Sia *lp* il link di tipo *Percorri* tra il battello `this` e la tratta *tr* percorsa, e sia $d = lp.mtPercorsi$ la distanza percorsa dal battello sulla tratta. Sia inoltre C l'insieme dei link di tipo *comprende* che coinvolgono la tratta *tr*. Il battello si trova in porto se: esiste un attracco sulla tratta la cui distanza dall'inizio della tratta è uguale ai metri percorsi dal battello sulla tratta, cioè se $\exists c \in C$ tale che $c.distanza = d$, e la lunghezza della banchina dell'attracco è almeno pari alla lunghezza dello scafo del battello, cioè
$c.attracco.lunghezzaBanchina >= this.lunghezza$
NOTA: si veda il metodo ausiliario `inPorto()` nella classe *BattelloFired*

Transizione: inNavigazione --> arrivato
percorri[arrivato]
Evento: percorri
Condizione: ([arrivato])
Sia lp il link di tipo $Percorri$ tra il battello `this` e la tratta tr percorsa, e sia $d = lp.mtPercorsi$ la distanza percorsa dal battello sulla tratta. Il battello è arrivato se i metri percorsi sulla tratta sono pari alla lunghezza della tratta stessa, cioè $d = tr.lunghezza()$.
NOTA: si veda il metodo ausiliario `arrivato()` nella classe $BattelloFired$

Transizione: inNavigazione -> inNavigazione
percorri[!arrivato && !inPorto]/percorri{dest:this}
Evento: percorri
Condizione: ([!arrivato && !inPorto])
Si vedano le definizioni delle condizioni $inPorto$ e $arrivato$ nelle specifiche delle transizioni precedenti
Azione:
pre: --
post: `nuovoevento = percorri{mitt=this, dest=this} AND`
sia $link$ il link di tipo $Percorre$ tale che $link.battello =$ `this`; allora rimpiazza il link esistente con un nuovo link $link'$ tale che $link'.mtPercorsi = link.mtPercorsi + 100$.
NOTA: si veda il metodo ausiliario `aggiornaMtPercorsi()` nella classe $BattelloFired$

Transizione: Attraccato -> inNavigazione
partenza/percorri{dest:this}
Evento: partenza
Condizione: --
Azione:
pre: --
post: `nuovoevento = percorri{mitt=this, dest=this} AND`
sia $link$ il link di tipo $Percorre$ tale che $link.battello =$ `this`; allora rimpiazza il link esistente con un nuovo link $link'$ tale che $link'.mtPercorsi = link.mtPercorsi + 100$.
NOTA: si veda il metodo ausiliario `aggiornaMtPercorsi()` nella classe $BattelloFired$

FineSpecifica

Attività di I/O

InizioSpecificaAttivitàAtomica InserisciDatiTratta
InserisciDatiTratta():(Tratta)
pre: --
post: Legge il nome e min/max profondità di una tratta, forniti in input dall'utente.
`result` è la tratta creata a partire dai dati inseriti.
FineSpecifica

InizioSpecificaAttivitàAtomica InserisciDatiBattello
InserisciDatiBattello():(Battello)
pre: --
post: Legge nome, numero di posti e lunghezza/profondità dello scafo di un battello, forniti in input dall'utente.
`result` è il battello creato a partire dai dati inseriti.
FineSpecifica

`InizioSpecificaAttivitàAtomica MostraErrore`
`MostraErrore():()`
`pre: --`
`post:` Visualizza un messaggio di errore che informa l'utente che la tratta non è navigabile dal battello.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica InserisciDatiAttracco`
`InserisciDatiAttracco():(RecordDatiAttracco)`
`pre: --`
`post:` Legge il nome di un attracco, la lunghezza della banchina e la distanza in metri dell'attracco stesso dal precedente (o dal porto di partenza per il primo attracco), forniti in input dall'utente.
`result` è l'oggetto *RecordDatiAttracco*[2] che contiene l'attracco creato a partire dai dati inseriti e la distanza inserita.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica ChiediSeAltroAttracco`
`ChiediSeAltroAttracco():(Bool)`
`pre: --`
`post:` Chiede all'utente se vuole aggiungere un altro attracco.
`result` è `true` in caso affermativo, `false` altrimenti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica VisualizzaSimulazione`
`VisualizzaSimulazione(b:Battello):()`
`pre: --`
`post:` Mostra la finestra di visualizzazione della simulazione del percorso del battello `b` sulla tratta.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica ChiediSeAltraSimulazione`
`ChiediSeAltraSimulazione():(Bool)`
`pre: --`
`post:` Chiede all'utente se vuole eseguire un'altra simulazione.
`result` è `true` in caso affermativo, `false` altrimenti.
`FineSpecifica`

Attività Atomiche

`InizioSpecificaAttivitàAtomica VerificaNavigabilita`
`VerificaNavigabilita(b:Battello, tr:Tratta):(Bool)`
`pre: --`
`post:` Verifica se la profondità minima della tratta `tr` è maggiore della profondità dello scafo del battello `b`
`result` è `true` se `tr.minProfondita > b.profondita`, `false` altrimenti.
`FineSpecifica`

`InizioSpecificaAttivitàAtomica PosizionaBattello`
`PosizionaBattello(b:Battello, tr:Tratta):()`
`pre: --`
`post:` Crea un link *link* di tipo *Percorre* tale che $link.mtPercorsi = 0$, $link.battello =$ `b` e $link.tratta =$ `tr`.
`FineSpecifica`

[2] *RecordDatiAttracco* è un record contente due campi: l'attracco e la distanza.

`InizioSpecificaAttivitàAtomica AggiungiAttracco`

`AggiungiAttracco(tr:Tratta, ra:RecordDatiAttracco):()`

`pre: --`

`post:` Crea un link *link* di tipo *Comprende* tale che *link.tratta* = `tr` e *link.attracco* = `ra.attracco`; *link.distanza* è pari a `ra.distanza` se la tratta `tr` non è coinvolta in alcun link di tipo *Comprende*; altrimenti, sia C l'insieme ordinato dei link di tipo *Comprende* che coinvolgono la tratta `tr`: *link.distanza* è pari a $lastElementOf(C).distanza$ + `ra.distanza`.

`FineSpecifica`

`InizioSpecificaAttivitàAtomica InizializzaSimulazione`

`InizializzaSimulazione(b:Battello):()`

`pre: --`

`post:` Inizializza l'`Environment`, inserendovi (come Listener) il battello `b`; successivamente, attiva i Listener.

`FineSpecifica`

Attività Composte

InizioSpecificaAttività AttivitaPrincipale

AttivitaPrincipale():()

Variabili Processo:
tratta: Tratta -- tratta corrente
battello: Battello -- battello corrente
navigabile: Bool -- tratta navigabile dal battello?
recordAttracco: RecordDatiAttracco -- record attracco corrente
altroAttracco: Bool -- inserire altro attracco?
altraSimulazione: Bool -- eseguire altra simulazione?

Inizio Processo:

```
do {
  InserisciDatiTratta():(tratta);
  InserisciDatiBattello():(battello);
  VerificaNavigabilita(battello, tratta):(navigabile);

  if (navigabile) {
    PosizionaBattello(battello, tratta):();

    do {
      InserisciDatiAttracco():(recordAttracco);
      AggiungiAttracco(tratta, recordAttracco):();
      ChiediSeAltroAttracco():(altroAttracco);
    } while(altroAttracco);

    InizializzaSimulazione(battello):();
    VisualizzaSimulazione(battello):();

    // Disattiva listener
    EsecuzioneEnvironment.disattivaListener();

    ChiediSeAltraSimulazione():(altraSimulazione);
  }
  else {
    mostraErrore():();
  }
} while(altraSimulazione);
```

FineSpecifica

Progetto

Responsabilità sulle Associazioni

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

Associazione	Classe	Ha Responsabilità
percorre	Battello	SI (M,O,R)
	Tratta	SI (M)
comprende	Tratta	SI (M,O,R)
	Attracco	SI (M)

Strutture di Dati

Rappresentiamo le collezioni omogenee di oggetti mediante le classi `Set` ed `HashSet` del Collection Framework di Java. Inoltre, rappresentiamo le collezioni omogenee ordinate di oggetti mediante le classi `List` e `ArrayList` 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.

Classe UML	Proprietà Immutabile
Battello	nome numeroPosti lunghezza profondita
Tratta	nomeFiume minProfondita maxProfondita
Attracco	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.

La finestra principale dell'applicazione deve essere simile a quella in Figura 4.

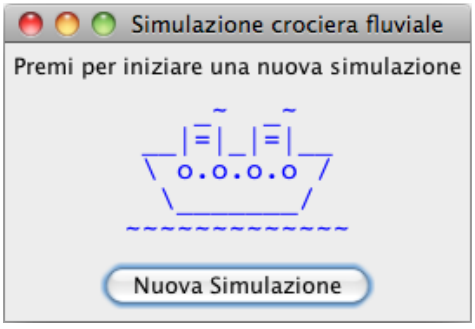

Figura 4: La finestra principale