Università degli Studi di Roma "La Sapienza"
Facoltà di Ingegneria – Corso di Laurea in Ingegneria Gestionale
**Corso di Progettazione del Software**
Proff. Toni Mancini e Monica Scannapieco

Progetto **E.20060411**

versione del 12 marzo 2007

Si vuole progettare un'applicazione che gestisce un'ampia gamma di pacchetti crociera per l'agenzia di viaggi *TravelToTheMoon*. Oltre al confezionamento dei pacchetti crociera ed alla gestione delle prenotazioni dei clienti, l'applicazione deve consentire al reparto marketing di *TravelToTheMoon* di fare delle indagini, cosí abilitando opportune strategie pubblicitarie.

Si richiede di effettuare le fasi di Analisi, Progetto, e Realizzazione del sistema in JAVA, utilizzando la metodologia illustrata nel corso.

## Requisiti

Delle crociere offerte dall'agenzia interessa il codice, le date di inizio e fine, e la nave utilizzata. Delle navi, che hanno un nome (ad es. "LoveBoat"), interessa il grado di comfort, espresso in un numero di stelle che può variare da 3 a 5, e il numero massimo di passeggeri che possono ospitare.

Ciascuna crociera consta di un itinerario caratterizzato da un nome (ad es. "Panorami d'Oriente") il quale prevede una sequenza –ordinata– di destinazioni. Di queste interessa il nome e il continente in cui si trovano. Gli itinerari fissano, oltre che l'ordine delle destinazioni da visitare, anche le relative date di arrivo e di partenza. Dato che, in generale, un itinerario può essere previsto da più di una crociera, le date di arrivo e partenza relative ad una destinazione vengono espresse come differenze rispetto la data di inizio della crociera stessa (ad es., l'itinerario "Panorami d'Oriente" prevede di raggiungere la destinazione $x$ alle 16:00 del quinto giorno di crociera, e di ripartire alle 12:00 del giorno successivo, il sesto).

Inoltre, le destinazioni sono caratterizzate da un insieme di posti da vedere durante eventuali escursioni organizzate. Questi ultimi sono caratterizzati dal nome, dalla descrizione, e dalla fascia oraria consigliata per le visite. Il sistema deve permettere di risalire ai posti da vedere in ogni singola destinazione.

L'agenzia classifica le crociere in *crociere di luna di miele* e *crociere per famiglia* (di queste ultime interessa conoscere se sono adatte o meno ai bambini), e le destinazioni in *romantiche* e *divertenti*. Si noti che possono esistere destinazioni che sono sia romantiche che divertenti. Per

venire incontro alle nuove tendenze delle giovani coppie, le crociere di luna di miele vengono ulteriormente classificate in *tradizionali* e *alternative*: sono definite tradizionali quelle che prevedono un numero di destinazioni romantiche maggiore o uguale al numero di destinazioni divertenti, alternative le altre.

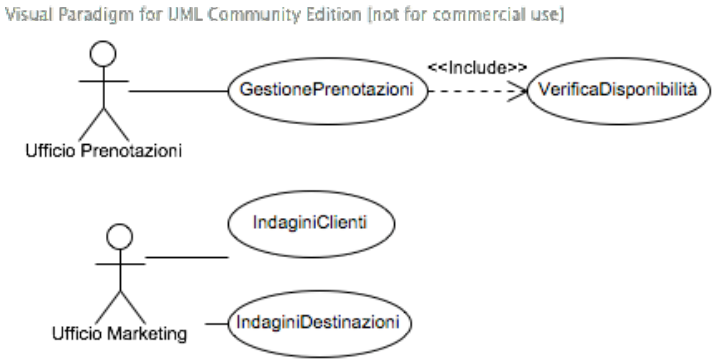
Infine, il sistema deve anche permettere di gestire le prenotazioni di crociere effettuate dai clienti. In particolare, dei clienti interessa nome, cognome, età ed indirizzo, mentre delle prenotazioni interessa l'istante di prenotazione, la crociera ed il numero di posti prenotati.

Le funzionalità richieste al sistema sono le seguenti:

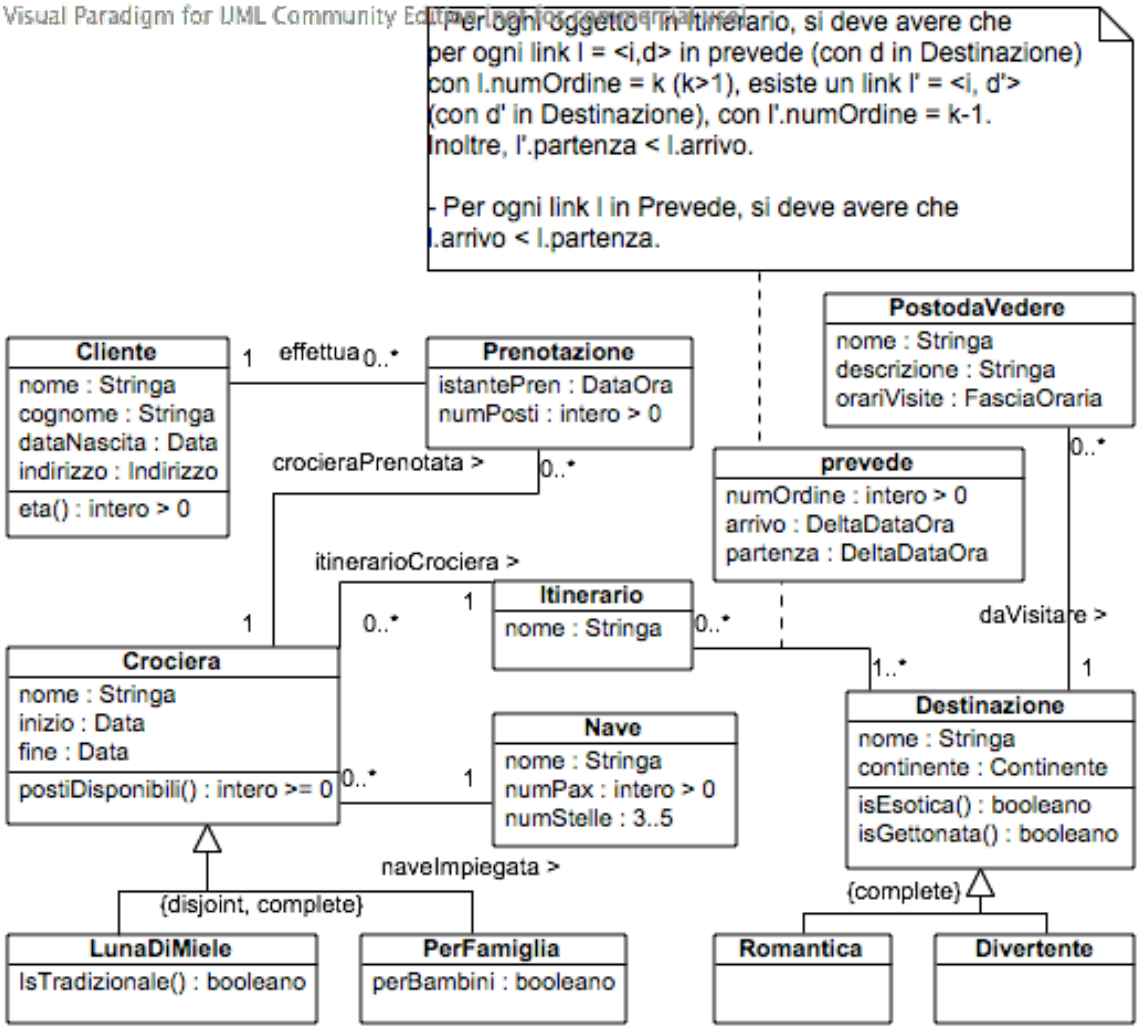
1. Dato un cliente che desidera prenotare un certo numero di posti per una crociera $c$, il personale dell'Ufficio Prenotazioni deve poter effettuare la relativa prenotazione. La richiesta di prenotazione deve essere rifiutata nel caso il numero di posti disponibili, alla data corrente, per la crociera $c$ non sia sufficiente.
2. Dato un insieme di clienti, l'Ufficio Marketing deve poter calcolare l'età media di quelli che hanno prenotato almeno una crociera che prevede una destinazione esotica (ovvero che si trova in un continente diverso dall'Europa).
3. Dato un insieme di destinazioni, l'Ufficio Marketing deve poter calcolare la percentuale di quelle *gettonate*. Una destinazione si dice gettonata se è stata raggiunta da almeno dieci crociere di luna di miele, oppure da almeno quindici crociere per famiglie nel corso degli ultimi due anni.

# 1 Fase di Analisi

## 1.1 Diagramma degli Use Case



## 1.2 Diagramma delle classi UML




## 1.3 Specifica degli use case

```
SpecificaUseCase GestionePrenotazioni
  prenota(cl: Cliente, cr: Crociera, nposti : intero > 0): Prenotazione
    pre:
      - adesso.data < cr.inizio, con adesso l'istanza del tipo DataOra relativa
        all'istante corrente;
```

```
      - VerificaDisponibilita.postiDisponibili(cr) >= nposti;

    post:
      Viene creato un oggetto p di classe Prenotazione, con:
        - p.istantePren = adesso;
        - p.numPosti = nposti.
      Vengono inoltre creati i seguenti link:
        - <cl, p> in effettua;
        - <cr, p> in crocieraPrenotata.

      result e' pari a p.
FineSpecifica

SpecificaUseCase VerificaDisponibilita
  postiDisponibili(cr: Crociera): intero >= 0
    pre: oggi < cr.inizio, con oggi l'istanza del tipo Data relativa
           all'istante corrente;
    post: result = cr.postiDisponibili();
FineSpecifica

SpecificaUseCase IndaginiClienti
  etaMediaEsotiche(C: Insieme(Cliente)): reale >= 0
    pre: |C| >= 1
    post:
      Detto C' il sottoinsieme di C dei clienti che hanno prenotato almeno una
      crociera che raggiunge una destinazione esotica, ovvero:

        C' = { c in C |
                    esiste un link <c,p> in effettua t.c.
                       esiste un link <p.crocieraPrenotata.Crociera.itinera-
                          rioCrociera.Itinerario, d> in
                            prevede tale che d.isEsotica()=true }

                          result = (Σ_{c∈C'} c.eta()) / |C'|

FineSpecifica

SpecificaUseCase IndaginiDestinazioni
  percentualeGettonate(D: Insieme(Destinazione)): reale in 0..100
    pre: |D| >= 1
```

```
    post:
      Detto D' il sottoinsieme di D composto da tutte e sole le
      destinazioni gettonate, ovvero:

        D' = { d in D | d.isGettonata()=true }.

      result = |D'|*100 / |D|.
FineSpecifica
```

## 1.4 Specifica delle classi

### La classe Crociera

```
SpecificaClasse Crociera
  postiDisponibili(): intero >= 0
    pre: adesso.data <= cr.inizio, con adesso l'istanza del tipo DataOra relativa
           all'istante corrente;
    post: Detto P l'insieme delle prenotazioni effettuate per la crociera this fino
         all'istante adesso:

         P = { p in Prenotazione | <this,p> in crocieraPrenotata e p.istantePren < adesso }

           result = this.naveImpiegata.Nave.numPax − Σ_{p∈P} p.numPosti

FineSpecifica
```

### La classe Destinazione

```
SpecificaClasse Destinazione
 isEsotica(): booleano
  pre: nessuna
  post: result e' pari a true se e solo se this.continente != EUROPA.

 isGettonata(): booleano
  pre: nessuna
  post:
    result e' pari a true se e solo se almeno una delle seguenti condizioni e' verificata:
    - |{ ldm in LunaDiMiele t.c.
            oggi.differenza(ldm.inizio, ANNI)<=2 e
```

```
            <ldm.itinerarioCrociera.Itinerario, this> in prevede }| >= 10;

    - |{ pf in PerFamiglie t.c.
            oggi.differenza(pf.inizio, ANNI)<=2 e
            <pf.itinerarioCrociera.Itinerario, this> in prevede }| >= 15

    con oggi l'istanza del tipo Data relativa alla data corrente.
FineSpecifica
```

**La classe Cliente**

```
SpecificaClasse Cliente
  eta(): intero > 0
   pre: nessuna
   post:
     detta oggi l'istanza del tipo Data relativa alla data corrente,
     result = parteInteraInferiore(oggi.differenza(this.dataNascita, ANNI)).
FineSpecifica
```

## 1.5 Specifica dei tipi di dato

```
SpecificaTipoDiDato Indirizzo
  attributi
    via: Stringa
    civico: intero > 0
    citta: Stringa
FineSpecifica

Continente = {AFRICA, AMERICA, ASIA, EUROPA, OCEANIA}.

SpecificaTipoDiDato FasciaOraria
  attributi
    da: Ora
    a: Ora
FineSpecifica

SpecificaTipoDiDato DeltaDataOra
  attributi
    giorno: intero > 0
```

```
    ora: Ora
  operazioni
    prima(altra DeltaDataOra): booleano
      pre: nessuna
      post: result e' pari a true se e solo se
         this.giorno < altra.giorno oppure
         this.giorno = altra.giorno e this.ora.prima(altra.ora)
FineSpecifica
```

# 2 Fase di Progetto

## 2.1 Corrispondenza tra tipi UML e tipi Java

| Tipo UML | Tipo Java | Note |
|---|---|---|
| Stringa | String | – |
| Indirizzo | Indirizzo | cf. spec. realizzativa |
| Data | Data | disponibile, usiamo vers. senza side-effect, con condiv. |
| DataOra | DataOra | disponibile, usiamo vers. senza side-effect, con condiv. |
| DeltaDataOra | DeltaDataOra | cf. spec. realizzativa |
| FasciaOraria | FasciaOraria | cf. spec. realizzativa |
| intero $>/\geq$ 0, 3..5 | int | Verifica ammissibilità sul lato server |
| Continente | int | 0=AFR., 1=AMER,, 2=ASIA, 3=EUR., 4=OC. – Verif. lato server |
| Insieme(...) | HashSet< ... > | Implementa l'interfaccia Set< ... > |

## 2.2 Ristrutturazione delle gerarchie is-a

La gerarchia *Crociera/LunaDiMiele/PerFamiglie* è disjoint e complete: la class Java `Crociera` sarà quindi `abstract`, mentre nessuna ristrutturazione è necessaria per le sottoclassi.

Al contrario la gerarchia *Destinazione/Romantica/Divertente* è complete ma non disjoint: andrebbe quindi ristrutturata creando la ulteriore sottoclasse Java `RomanticaDivertente`. La class `Destinazione` sarà abstract (perché la gerarchia è complete). Tuttavia, in questo caso, siamo in un caso molto particolare: le sottoclassi non hanno alcun attributo e non sono coinvolte in alcuna associazione. Per semplicità ristrutturiamo quindi la gerarchia in modo diverso: eliminiamo le sottoclassi `Romantica` e `Divertente`, e le sostituiamo con due attributi di tipo `boolean` nella classe `Destinazione` (che non sarà quindi abstract). Il fatto che la gerarchia originaria è complete, impone poi il vincolo che, per ogni oggetto di classe `Destinazione`, il valore di almeno uno di tali attributi debba essere true.

## 2.3 Specifica realizzativa delle strutture dati

```
SpecificaStrutturaDati Indirizzo
  attributi
   +via: String
   +civico: int
   +citta: String
  operazioni
    --
  schema realizzativo
    senza side-effect, con condiv.
  controllo uguaglianza
    campo a campo
FineSpecifica

SpecificaStrutturaDati FasciaOraria
  attributi
   +da: Ora
   +a: Ora
FineSpecifica

SpecificaStrutturaDati DeltaDataOra
  attributi
   +giorno: int
   +ora: Ora
  operazioni
   +prima(altra DeltaDataOra): boolean
      pre: nessuna
      algoritmo:
        se this.giorno < altra.giorno oppure
             this.giorno = altra.giorno e this.ora.prima(altra.ora)
               allora ritorna true;
        altrimenti ritorna false.
FineSpecifica
```

## 2.4 Specifica realizzativa delle classi

### La classe Crociera

```
SpecificaRealizzativaClasse Crociera
```

```
 +postiDisponibili(): int
    pre: cf. spec. concettuale
    algoritmo:
        adesso = data/ora corrente;
        result = this.naveImpiegata.Nave.numPax;
        per ogni 'l' in this.crocieraPrenotata {
          se l.Prenotazione.dataPren.primaDi(adesso)
             result -= l.Prenotazione.numPosti;
        }
        ritorna result;
FineSpecifica
```

**La classe Destinazione**

```
SpecificaRealizzativaClasse Destinazione
+isEsotica(): boolean
  pre: nessuna
  algoritmo: se this.continente != 3 (EUROPA) ritorna true;
           altrimenti ritorna false.

+isGettonata(): boolean
  pre: nessuna
  algoritmo:
     oggi = data corrente;
     ldm = 0;
     pf = 0;
     per ogni 'l' in this.prevede {
       it = l.Itinerario;
       per ogni 'll' in it.itinerarioCrociera {
         c = ll.Crociera;
         se oggi.differenza(c.inizio, ANNI)<=2 allora {
            se c e' di classe LunaDiMiele, allora ldm++;
            altrimenti pf++;
         }
       }
     }
     se ldm >= 10 oppure pf >= 15, allora ritorna true;
     altrimenti ritorna false.
FineSpecifica
```

**La classe Cliente**

```
SpecificaClasse Cliente
 +eta(): int
   pre: nessuna
   algoritmo:
     oggi = data corrente;
     ritorna parteInteraInferiore(oggi.differenza(this.dataNascita, ANNI)).
FineSpecifica
```

## 2.5 Specifica realizzativa degli use case

```
SpecificaRealizzativaUseCase GestionePrenotazioni
 +prenota(cl: Cliente, cr: Crociera, nposti : int): Prenotazione
    pre: adesso = data/ora corrente;
      deve essere:
      - adesso.data.prima(cr.inizio);
      - VerificaDisponibilita.postiDisponibili(cr) >= nposti;

    algoritmo:
      Crea un oggetto p di classe Prenotazione, con:
        - p.istantePren = adesso;
        - p.numPosti = nposti.
      Crea i link:
        - <cl, p> in effettua;
        - <cr, p> in crocieraPrenotata.

      ritorna p;
FineSpecifica

SpecificaUseCase VerificaDisponibilita
 +postiDisponibili(cr: Crociera): int
    pre: detto oggi = data corrente;
            deve essere oggi.prima(cr.inizio.data)=true
    algoritmo: ritorna cr.postiDisponibili();
FineSpecifica

SpecificaUseCase IndaginiClienti
 +etaMediaEsotiche(C: Set<Cliente>): double
```

```
    pre: |C| >= 1
    algoritmo:
      somma = 0;
      num=0;
      per ogni 'c' in C {
         faMedia=false;
         per ogni 'l' in c.effettua {
           it = l.Prenotazione.crocieraPrenotata.Crociera.itinerarioCrociera.Itinerario;
           per ogni 'll' in it.prevede {
             d = ll.Destinazione;
             se d.isEsotica()=true, allora faMedia=true;
           }
         }
         se faMedia=true allora {
            somma += c.eta();
            num++;
      }
      ritorna somma/num;
FineSpecifica

SpecificaUseCase IndaginiDestinazioni
 +percentualeGettonate(D: Set<Destinazione>): double
    pre: |D| >= 1
    post:
      cont = 0;
      per ogni d in D {
        se d.isGettonata()=true allora cont++;
      }
      ritorna cont*100 / |D|;
FineSpecifica
```

## 2.6 Progetto dei diagrammi degli stati

Non sono stati definiti diagrammi degli stati in fase di Analisi.

## 2.7 Responsabilità sulle associazioni

Dai requisiti, dalla specifica delle operazioni di classi e di use case, e delle molteplicità nel diagramma delle classi emerge che:

| Associazione | Classe | Ha resp? | Motivo |
|---|---|---|---|
| effettua | Cliente | SI' | u.c., op. 2 |
| | Prenotazione | SI' | 1..1 |
| crocieraPrenotata | Prenotazione | SI' | 1..1 |
| | Crociera | SI' | u.c., op. 1 |
| itinerarioCrociera | Crociera | SI' | 1..1 |
| | Itinerario | SI' | u.c., op. 3 |
| naveImpiegata | Crociera | SI' | 1..1 |
| | Nave | NO | – |
| prevede | Itinerario | SI' | 1..* |
| | Destinazione | SI' | u.c., op. 2&3 |
| daVisitare | PostoDaVedere | SI' | 1..1 |
| | Destinazione | SI' | requisiti (il sistema deve permettere...) |

## 2.8 Vincoli sull'evoluzione delle proprietà mutabili

Tutte le proprietà mutabili possono variare arbitrariamente, con le seguenti eccezioni:

- I link di associazioni *effettua* e *crocieraPrenotata* possono essere eliminati soltanto prima dell'inizio della crociera relativa.

## 2.9 Diagramma delle classi realizzativo