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 **PI.20050411**, passo **P.1**

versione del 16 marzo 2007

Il Comune di Roma, d'intesa con il consorzio Metrebus, ha avviato nel 2005 l'erogazione del servizio di *car sharing* in città[1] (chiamato RomaCarSharing). Il servizio di car sharing permette ai soci di noleggiare, presso uno dei garage convenzionati, disseminati nel territorio, un'autovettura anche solo per poche ore, pagando l'uso effettivo del veicolo (ovvero un costo fisso, più un costo orario, mentre il carburante è compreso). I vantaggi sono molteplici, soprattutto per coloro che non usano frequentemente l'auto, potendo evitare i costi (assicurazione, bollo, manutenzione) derivanti dal possesso di un'auto privata.

Si vuole progettare un'applicazione che permetta di gestire alcune informazioni sul servizio di car sharing, relativamente ai soci e alle auto noleggiate.

Si richiede di effettuare la fase di Progetto del sistema complessivo, utilizzando la metodologia illustrata nel corso.

1 Fase di Progetto

1.1 Corrispondenza tra tipi UML e tipi Java

[1] Per ora, in via sperimentale, solo nel III Municipio, dal 2006 sarà gradualmente esteso a tutto il territorio cittadino. Per info: `www.comune.roma.it`.

Tipo UML	Tipo Java	Note
Stringa	String	–
Stringa(n)	String	Verifica cond. ammiss. lato server
DataOra	DataOra	usiamo vers. senza side-effect con condiv. – disponibile
intero > 0, 0..10, 0..100	int	Verifica cond. ammiss. lato server
booleano	boolean	–
Insieme(...)	HashSet< ... >	Implementa l'interf. Set< ... >
reale $> / \geq 0$	double	Verifica cond. ammiss. lato server
Indirizzo	Indirizzo	cf. spec. realizzativa
CoordBancarie	CoordBancarie	cf. spec. realizzativa
Stringa {1,*}	HashSet< *String* >	–

1.2 Specifica realizzativa delle strutture dati

La struttura dati Indirizzo

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

La struttura dati CoordBancarie

```
SpecificaStrutturaDati CoordBancarie
  attributi
    abi: int
    cab: int
    cc: int
  schema realizzativo
    senza side-effect, con condiv.
  controllo uguaglianza
    campo a campo
```

```
FineSpecifica
```

1.3 Specifica realizzativa degli use-case

Use case AutoPreferite

```
SpecificaUseCase AutoPreferite
 +autoPreferite(s : Socio) : Set<Auto>
    pre: nessuna
    algoritmo: ritorna s.autoPreferite().
FineSpecifica
```

Use case ImportoDovuto

```
SpecificaUseCase ImportoDovuto
 +importoDovuto(s : Socio, da : DataOra, a : DataOra) : double
    pre: Quelle di s.importoDovuto().
    algoritmo: ritorna s.importoDovuto(da, a).
FineSpecifica
```

Use case VerificheSuConvenzioni

```
SpecificaUseCase VerificheSuConvenzioni
 +societaConConvPiuVantaggiose(c : Categoria) : Set<Societa>
    pre: nessuna
    algoritmo:
       CCmax = insieme vuoto di oggetti di classe Categoria;
       scontoMax = 0;
       per ogni 'l' in c.convenzioneRelativaA {
          cc = l.ConvenzioneCategorie;
          se cc.sconto = scontoMax {
             CCmax = CCmax U {cc};
          }
          altrimenti, se cc.sconto > scontoMax {
             CCmax = {cc};
             scontoMax = cc.sconto;
          }
       }
```

```
      result = insieme vuoto di oggetti di classe Societa
      per ogni cc in CCmax {
         per ogni 'l' in cc.haConvenzione {
            result = result U {l.Societa}
         }
      }
      ritorna result;

  +almenoUnaConvenzioneEco(I : Set<Societa>) : Set<Societa>
    pre: nessuna
    algoritmo:
      result = insieme vuoto di oggetti di classe Societa;
      per ogni 's' in I {
          per ogni 'l' in s.haConvenzione {
             se l.Convenzione e' di classe ConvenzioneEcocompatibili
                allora {
                  result = result U {s};
                  esci dal ciclo ''per ogni l'';
                }
          }
      }
      ritorna result;

FineSpecifica
```

Use case PrenotazioneAuto

```
SpecificaUseCase PrenotazioneAuto
 +prenotaAuto(s:Socio, auto:Auto, inizio:DataOra, da:Garage,
                        fine:DataOra, a:Garage) : Noleggio

    pre:
         - inizio.prima(fine) oppure inizio = fine;
         - adesso.prima(inizio), con 'adesso' l'istanza del tipo DataOra che
               rappresenta l'istante corrente,
         - auto.disponibile(inizio, fine) = true, e
         - auto.siTrovaIn(inizio, da) = true.

    algoritmo:
```

```
        result = nuovo oggetto di classe Noleggio con:
                result.inizio = inizio,
                result.fine = fine.
        crea i seguenti nuovi link:
                <result, s> di associazione clienteNoleggio,
                <result,auto> di associazione autoNoleggiata,
                <result, da> di associazione presaInConsegna,
                <result,a> di associazione rilascio.
        ritorna result;
FineSpecifica
```

Use case ClasseDiRischio

```
SpecificaUseCase ClasseDiRischio
 +classeDiRischio(s:Socio) : int
    pre: nessuna
    algoritmo: ritorna s.classeDiRischio().
FineSpecifica
```

Use case UltimoUtente

```
SpecificaUseCase UltimoUtente
 +ultimoUtente(a:Auto, momento:DataOra) : Socio
    pre: Quelle di a.ultimoUtente(momento).
    algoritmo: ritorna a.ultimoUtente(momento).
FineSpecifica
```

1.4 Specifica realizzativa delle classi

La classe Socio

```
SpecificaClasse Socio (abstract)
  +classeDiRischio() : int
    pre: nessuna
    algoritmo:
        adesso = istante corrente;
        numSinistri = 0;
        per ogni 'l' in this.clienteNoleggio {
```

```
          n = l.Noleggio;
          se adesso.differenza(n.inizio, ANNI) <= 3 allora {
              per ogni 'ls' in n.sinistroRelativoA {
                 se non e' vero che
                     ( ls.Sinistro e' di classe SinistroConControparte e
                       s.haColpa = false )
                        allora numSinistri++;
              }
          }
          ritorna min(numSinistri, 10);

  +abstract clienteAbituale() : boolean

  #clienteAbituale(x: int) : boolean
    pre: nessuna
    algoritmo:
        num = 0;
        per ogni 'l' in this.clienteNoleggio {
           n = l.Noleggio;
           se adesso.differenza(n.inizio, MESI) <= 12 allora num++;
        }
        ritorna (num >= x);

  +autoPreferite() : Set<Auto>
     pre: nessuna
     algoritmo:
       N = insieme vuoto di oggetti di classe Noleggio;
       per ogni 'l' in this.clienteNoleggio {
           N = N U {l.Noleggio};
       }
       A = insieme vuoto di oggetti di classe Auto;
       per ogni n in N {
         per ogni 'l' in n.autoNoleggiata {
            A = A U {l.Auto};
         }
       }
       Amax = insieme vuoto di oggetti di classe Auto;
       max = 0;
       per ogni 'a' in A {
         quanteVolteQuesta = 0;
```

```
        per ogni 'l' in a.autoNoleggiata {
           se l.Noleggio.clienteNoleggio.Cliente = this, allora quanteVolteQuesta++;
        }
        // quanteVolteQuesta e' pari al numero di volte che this ha noleggiato 'a'
        se quanteVolteQuesta = max, allora Amax = Amax U {a}
        altrimenti, se quanteVolteQuesta > max, allora {
           Amax = {a};
           max = quanteVolteQuesta;
        }
      }
      ritorna Amax;

  +importoDovuto(da : DataOra, a : DataOra) : double
     pre:
       -da.prima(a) oppure a = da;
       - a.prima(adesso) oppure a = adesso, con 'adesso' l'istante corrente.

     algoritmo:
        N = insieme vuoto di oggetti di classe Noleggio;
        per ogni 'l' in this.clienteNoleggio {
           n = l.Noleggio;
           se (da.prima(n.inizio) oppure da = n.inizio) e
                (n.inizio.prima(a)) allora
                      N = N U {n};
        }
        importoBase = 0;
        per ogni n in N {
           importoBase = importoBase + n.prezzo();
        }
        se this.clienteAbituale()=true, ritorna importoBase * 0.85
        altrimenti ritorna importoBase.

FineSpecifica
```

La classe Privato

```
SpecificaClasse Privato is-a Socio
 +clienteAbituale() : boolean
    pre: nessuna
```

```
    algoritmo: ritorna super.clienteAbituale(20).
FineSpecifica
```

La classe DipendenteDiSocieta

```
SpecificaClasse DipendenteDiSocieta is-a Socio
 +clienteAbituale() : boolean
    pre: nessuna
    algoritmo: ritorna super.clienteAbituale(40).
FineSpecifica
```

La classe Noleggio

```
SpecificaClasse Noleggio
 +prezzo() : double
    pre: nessuna
    algoritmo:
         c = this.autoNoleggiata.Auto.diCategoria.Categoria.
         numore = parteInteraSup(this.fine.differenza(this.inizio, ORE))
         ritorna c.prezzoBase + c.prezzoOrario*numore;
FineSpecifica
```

La classe Auto

```
SpecificaClasse Auto
  +disponibile(inizio : DataOra, fine : DataOra) : boolean
    pre: inizio.prima(fine) oppure inizio = fine
    algoritmo:
       se le precondizioni di this.ultimoNoleggio(fine) non sono rispettate
          allora ritorna true.

      Altrimenti:
         n = this.ultimoNoleggio(fine);
         se n.fine.prima(inizio) oppure fine = inizio, allora ritorna true;
         altrimenti ritorna false;

  -ultimoNoleggio(momento : DataOra) : Noleggio
        (Nota: posso dichiararla privata, perche' di supporto, nonostante l'abbia
```

```
        sintetizzata in fase di Analisi)
   pre:
      n_max, al termine del ciclo 'per ogni' dell'algoritmo, non deve essere null.

   algoritmo:
      n_max = null;
      per ogni 'l' in this.autoNoleggiata {
          se (n_max = null   oppure   n_max.inizio.prima(l.Noleggio.inizio)) e
               (l.Noleggio.inizio.prima(momento) oppure l.Noleggio.inizio = momento)
             allora n_max = l.Noleggio;
      }
      ritorna n_max.

  +siTrovaIn(momento : DataOra, garage : Garage) : boolean
    pre: nessuna
    algoritmo:
      Se le precondizioni di this.ultimoNoleggio(momento) non sono
      rispettate, ritorna true.

      Altrimenti:
         n = this.ultimoNoleggio(momento);
         ritorna ( (n.fine.prima(momento) oppure n.fine = momento) e
                       (n.rilascio.Garage = garage) ).

  +ultimoUtente(momento : DataOra) : Socio
     pre: Le precondizioni di this.ultimoNoleggio(momento) sono rispettate.
     algoritmo: ritorna this.ultimoNoleggio(momento).clienteNoleggio.Socio.

FineSpecifica
```

1.5 Progetto dei diagrammi degli stati

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

1.6 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
clienteNoleggio	Socio	SI	*Socio.autoPreferite()* *Socio.importoDovuto()* *Socio.classeDiRischio()* *Socio.clienteAbituale()*
	Noleggio	SI	vincolo 1..1 *Auto.ultimoUtente()*
presaInConsegna	Noleggio	SI	vincolo 1..1
	Garage	NO	–
rilascio	Noleggio	SI	vincolo 1..1 *Auto.siTrovaIn()*
	Garage	NO	–
autoNoleggiata	Noleggio	SI	vincolo 1..1 *Socio.autoPreferite()* *Noleggio.prezzo()*
	Auto	SI	*Auto.disponibile()* *Auto.ultimoNoleggio()* *Auto.siTrovaIn()* *Auto.ultimoUtente()*
diCategoria	Auto	SI	vincolo 1..1 *Noleggio.prezzo()*
	Categoria	NO	–
lavoraIn	DipSocieta	SI	vincolo 1..1
	Societa	NO	–
haConvenzione	Societa	SI	op. u.c. *almenoUnaConvEco()*
	Convenzione	SI	op. u.c. *societaConConvPiuVant()*
convRelativaA	ConvCategorie	SI	vincolo 1..*
	Categoria	SI	op. u.c. *societaConConvPiuVant()*
sinistroRelativoA	Noleggio	SI	*Socio.classeDiRischio()*
	Sinistro	SI	vincolo 1..1

1.7 Vincoli sull'evoluzione delle proprietà mutabili

Tutte le proprietà mutabili possono variare arbitrariamente, tranne quelle contrassegnate dai commenti del diagramma delle classi realizzativo, per le quali sono espressi vincoli.

1.8 Diagramma delle classi realizzativo