Fondamenti di informatica L-B: esercitazione 04

Nome: Andrea Urbini urbo83@tiscali.it

Matricola: 167064

Descrizione problema

L'esercitazione chiede la realizzazione di un prototipo di biglietteria elettronica, per permettere la prenotazione e acquisto di biglietti per determinati spettacoli. Il teatro conta di 33 posti numerati (da 1 a 33). I servizi messi a disposizione da una biglietteria elettronica sono di due categorie, per utenti e per gestori. Il gestore deve avere la possibilità di inserire gli spettacoli disponibili; ad un utente invece devono essere forniti servizi per prenotare posti relativi ad uno spettacolo e, dopo aver prenotato, procedere all'acquisto dei biglietti. Quindi per il gestore devono essere forniti come servizi:

- Registrazione di nuovo spettacolo, caratterizzato da un nome;

Per gli utenti devono esser forniti come servizi:

- Prenotazione di uno o più posti per la visione di uno spettacolo. Uno spettacolo è caratterizzato da un nome (stringa). Se l'operazione va a buon fine all'utente viene rilasciata una nota di prenotazione, in altre parole qualcosa che identifichi la prenotazione effettuata, mantenendone le informazioni (nome dello spettacolo, posti prenotati);
- Acquisto dei biglietti, data una prenotazione. L'acquisto può esser valida, ottenuta in precedenza con il servizio di prenotazione.

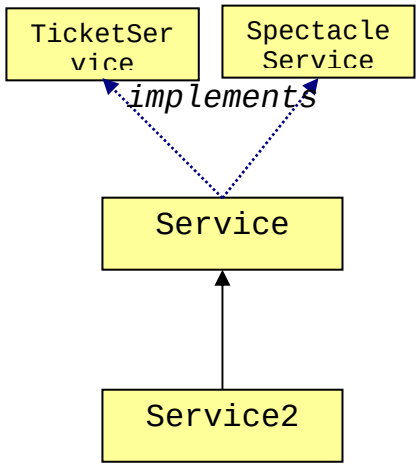
Ogni qualvolta avviene una prenotazione o un acquisto, la biglietteria deve tener nota dei fatti scrivendo un file di testo (*service.txt*) contendo le informazioni relative alle prenotazioni e agli acquisti.

Analisi

La software house ci ha fornito due interfacce relative alla soluzione del problema.
La prima (*TicketService*) contiene i due metodi necessari all'utente per interagire con il teatro:

- *bookTickets* – permette all'utente di prenotare i posti per uno spettacolo; viene rilasciata una ricevuta nel formato *BookingNote*.
- *BuyTickets* – presentando la ricevuta ottenuta all'atto di prenotazione si possono acquistare i biglietti.

La seconda interfaccia (*SpectacleService*) contiene il metodo necessario al gestore per registrare i nuovi spettacoli in una lista. Il metodo è chiamato: *registerSpectacle*.
Le due interfacce possono essere implementate da un'unica classe, chiamata *Service*, che rappresenta l'astrazione vera e propria della biglietteria del teatro, infatti i suoi metodi rispecchiano tutte (o quasi) le funzioni richieste ad una biglietteria.



Analizziamo le tre funzioni svolte da una biglietteria:

1. **registrazione di uno spettacolo** in una lista: lo spettacolo va inserito in una lista contenente tutti gli spettacoli in programmazione nel teatro.
2. **prenotazione dei biglietti:** all'atto della prenotazione si deve specificare il nome dello spettacolo e i numeri di posto che si vogliono prenotare. Una prenotazione va a buon fine se lo spettacolo richiesto è nella lista e se i posti da prenotare sono ancora liberi. In questo caso viene fornita una ricevuta caratterizzata dal nome dello spettacolo, dal numero di posti prenotati e da un numero univoco di identificazione.
3. **acquisto dei biglietti:** avviene tramite la presentazione della ricevuta di prenotazione; inoltre viene controllato che la prenotazione non sia già stata utilizzata per acquistare i posti.

Come si può notare, il sistema si deve ricordare dei posti prenotati e di quelli acquistati per ogni spettacolo, quindi queste informazioni vanno immagazzinate in una nuova identità (un oggetto) caratterizzata da una stringa *nome* e

contenente due *array* che mantengono le informazioni sui posti prenotati e acquistati; In particolare ogni vettore contiene il numero identificativo del posto se è, rispettivamente, prenotato o acquistato.
La creazione e la gestione di questo oggetto va assegnata ad una classe (*Spectacle*).

Per gestire la input/output di file, si crea una classe derivata da *Service* (*Service2*) in grado di immagazzinare su file di testo le informazioni nel seguente formato:

Messaggio di prenotazione	Sono stati prenotati XX posti per lo spettacolo YYYYY. Prenotazione: ZZZZ
Messaggio d'acquisto	Sono stati acquistati i biglietti relativi alla prenotazione ZZZZ

In un primo momento emerge quindi la necessità di progettare due entità che realizzino le strutture di tipi di dato di tipo astratto desiderate. Successivamente va creata una nuova entità, derivata da una delle precedenti in grado di realizzare il nuovo servizio richiesto.

Progetto

1. **La classe *Spectacle***
   L'oggetto creato da questo ADT è caratterizzato da un nome, specificato in fase di creazione, e da due *array* contenenti i numeri dei posti prenotati e acquistati; Gli *array*, inizialmente di lunghezza nulla, devono poter essere modificati dall'esterno, per questo devono essere introdotti dei metodi che li "estraggono" dall'oggetto *Spectacle* e altri metodi che li "inseriscono".

   I campi privati della classe sono i seguenti:

Nome	Tipo	Cosa contiene
name	String	Nome dello spettacolo
bookedSeats	int[]	Numeri dei posti prenotati
boughtSeats	int[]	Numeri dei posti acquistati

   Questa classe deve dunque avere un costruttore…

   public Spectacle(String name)

   …dei selettori…

Signature	Cosa fa
public String getName()	Restituisce la stringa *name*
public int[] getBookedSeats()	Restituisce l'*array bookedSeats* contenente i numeri dei posti prenotati.
public int[] getBoughtSeats()	Restituisce l'*array boughtSeats* contenente i numeri dei posti acquistati.

   …dei modificatori…

Signature	Cosa fa
public void setBookedSeats(int[] seats)	Assegna al riferimento *bookedSeats* l'array di interi *seats.*
public void setBoughtSeats(int[] seats)	Assegna al riferimento *boughtSeats* l'array di interi *seats.*

2. **La classe *Service***
   Questa classe implementa le due interfacce fornite: *TicketService* e *SpectacleService.* Il costruttore, crea un vettore (utilizzando la classe *java.util.Vector*) che andrà a contenere gli spettacoli registrati. Esso viene assegnato al campo *SpectaclesList* (ovviamente di tipo *Vector*) dichiarato *protected* per far fronte alla necessità di estendere questa classe (vedi *Service2*).
   La classe deve implementare tutti i metodi dichiarati nelle due interfacce, in particolare:

   public void registerSpectacle(String name)
   questo metodo crea un nuovo oggetto di tipo *Spectacle* definito dalla stringa *name* e lo aggiunge al vettore *SpectaclesList*.

   public BookingNote bookTickets(String spectacleName, int[] seats)

questo metodo permette di prenotare i posti *seats* per lo spettacolo identificato da *spectacleName*. Se la prenotazione ha successo viene rilasciata la ricevuta: un oggetto di tipo *BookingNote*. La validità della prenotazione viene certificata tramite due controlli implementati nel seguente modo:

- Ricerca dello spettacolo richiesto nella *SpectaclesList*: si confronta la stringa *spectacleName* con le stringhe identificative (ottenute con *getName*) di tutti gli elementi della *SpectaclesList*. Se lo spettacolo non viene trovato viene restituito *null*. Se lo spettacolo è stato trovato si estrae dalla SpectaclesList e si assegna alla variabile (di tipo *Spectacle*) *show*.
- Verifica della disponibilità dei posti (non devono già essere stati occupati): si confrontano tutti gli elementi dell'*array seats* di ingresso con quelli dell'array *bookedSeats* (ottenuto con il metodo *getBookedSeats*) contenuto in *show*. Se almeno uno dei posti è già stato prenotato la prenotazione fallisce e viene restituito *null*. Se tutti i posti sono liberi, viene concatenato a *bookedSeats* l'*array seats* e "passato" all'oggetto *show* tramite il metodo *setBookedSeats*. Lo spettacolo *show* viene poi reinserito dentro la *SpectaclesList*. Infine viene creata la ricevuta in formato BookingNote "passando" al relativo costruttore il nome dello spettacolo e l'*array seats*.

public boolean buyTickets(BookingNote note)

questo metodo permette di acquistare i posti prenotati in precedenza. Se l'acquisto ha successo viene ritornata la conferma di tipo *boolean*. L'acquisto ha successo se il nome dello spettacolo è nella lista e se i posti non sono già stati acquistati. Per effettuare questi controlli si procede nel seguente modo:

- Ricerca dello spettacolo richiesto nella *SpectaclesList*: si confronta la stringa corrispondente al nome dello spettacolo, estratta dalla *note* tramite il metodo *getName*, con le stringhe identificative (ottenute con *getName*) di tutti gli elementi della *SpectaclesList*. Se lo spettacolo non viene trovato viene restituito *false*. Se lo spettacolo viene stato trovato, si estrae dalla SpectaclesList e si assegna alla variabile (di tipo *Spectacle*) *show*.
- Verifica della validità della prenotazione (non deve essere già stata utilizzata per acquistare i posti): si confrontano tutti gli elementi dell'*array seats* contenuto nella *note* con quelli dell'array *boughtSeats* (ottenuto con il metodo *getBoughtSeats*) contenuto in *show*. Se almeno uno dei posti è già stato acquistato l'acquisto fallisce e viene restituito *false*. Se i posti non sono ancora stati acquistati, viene concatenato a *boughtSeats* l'*array seats* e "passato" all'oggetto *show* tramite il metodo *setBoughtSeats*. Lo spettacolo *show* viene poi reinserito dentro la *SpectaclesList*. Infine viene restituito il valore *true* che certifica la riuscita dell'acquisto.

3. **La classe *Service2***

Questa classe estende la *Service* aggiungendo la possibilità di mantenere traccia, su file di testo, di tutte le operazioni di prenotazione e acquisto effettuate dagli utenti. Quando una prenotazione o un acquisto hanno successo, sul file deve comparire una stringa indicante cosa è successo. Occorre re-implemetare i due metodi della classe Service riservati all'utente: *bookTickets* e *buyTickets* aggiungendo loro la capacità di scrivere un file. Per entrambi si richiama il metodo corrispondente nella superclasse e si controlla il tipo di dato restituito. Nel caso di una prenotazione, se viene restituito un oggetto *BookingNote* allora si crea la stringa *messaggio*, contenete informazioni sulla prenotazione, e la si passa al metodo *makeFile*. Se un acquisto riesce viene restituito *true* e si può passare alla creazione della stringa *messaggio* da passare al metodo *makeFile*.

private void makeFile(String messaggio)

Questo metodo, tramite le funzioni del package *java.io*, che deve importato all'inizio del codice, scrive sul *file service.txt* la stringa messaggio. Il tutto viene eseguito nel seguente modo:
Si crea una variabile di nome *file* e di tipo *FileWriter*; successivamente si prova ad assegnargli un oggetto di tipo *FileWriter* corrispondente a *service.txt* a cui è stata abilitata la funzione *append* (le stringhe successive vengono inserite in coda al file). Se viene lanciata un'eccezione di tipo *IOException* allora viene stampato su standard output il messaggio d'errore "Impossibile aprire il file" e il programma termina.
La fase di scrittura vera e propria inizia con il comando *write(messaggio)* che inserisce dentro a *file* la stringa messaggio, poi viene chiuso lo stream. Se viene lanciata una eccezione di tipo *IOException* viene stampato su standard output il messaggio d'errore "Errore di output" e il programma termina.

Implementazione

Listato corrispondente alla classe Spectacle.class

```
package es04;
```

```
/**
 * Questa classe rappresenta l'astrazione di uno spettacolo identificato da
 * un nome. Un oggetto Spectacle contiene anche informazioni sui posti
 * prenotati e occupati.
 * @author Andrea Urbini
 */

public class Spectacle{

	private String name;
	private int[] bookedSeats,boughtSeats;

	/**
	 * Crea un oggetto di tipo Spectacle caratterizzato da una stringa name.
	 * @param name nome dello spettacolo.
	 */

	public Spectacle(String name){
		this.name=name;
		bookedSeats=new int[0];
		boughtSeats=new int[0];
	}

	/**
	 * Fornisce una stringa contenente nome dello spettacolo.
	 * @return il nome dello spettacolo.
	 */

	public String getName(){
		return name;
	}

	/**
	 * Fornisce l'array dei posti che sono stati occupati
	 * @return posti occupati.
	 */

	public int[] getBookedSeats(){
		return bookedSeats;
	}

	/**
	 * Fornisce l'array dei posti che sono stati acquistati
	 * @return posti acquistati.
	 */

	public int[] getBoughtSeats(){
		return boughtSeats;
	}

	/**
	 * Aggiorna l'array dei posti prenotati
	 * @param seats array dei posti da occupare.
	 */

	public void setBookedSeats(int[] seats){
		bookedSeats=seats;
	}

	/**
	 * Aggiorna l'array dei posti acquistati
	 * @param seats array dei posti da acquistare.
	 */

	public void setBoughtSeats(int[] seats){
		boughtSeats=seats;
	}

}
```

Listato corrispondente alla classe Service.class

```
package es04;

import java.util.Vector;

/**
```

```
 * Questa classe rappresenta un prototipo di biglietteria elettronica. Implementa
 * le interfacce TicketService e SpectacleService.
 * @author Andrea Urbini
 */

public class Service implements TicketService,SpectacleService{

	/** Lista degli spettacoli in programmazione.*/

	protected Vector spectaclesList;

	/**
	 * Costruisce un oggetto di tipo Service: una astrazione di biglietteria.
	 */

	public Service(){
		spectaclesList=new Vector();
	}

	/**
	 * Registra uno spettacolo. Viene creato un nuovo oggetto Spectacle definito
	 * dalla stringa name e inserito nel vettore spectaclesList.
	 * @param name nome dello spettacolo da registrare.
	 */

	public void registerSpectacle(String name){
		spectaclesList.add(new Spectacle(name));
	}

	/**
	 * Servizio per prenotare i posti di uno spettacolo. Si specificano le
	 * informazioni sullo spettacolo da vedere e sui i posti da prenotare;
	 * il servizio prenota i posti e restituisce una prenotazione contenente
	 * tali informazioni.
	 * @param spectacleName nome dello spettacolo da ricercare
	 * @param seats array contenente i posti che si intendono prenotare.
	 * @return Se l'operazione ha successo viene restituita una prenotazione,
	 * al contrario se l'operazione non riesce viene restituito il valore null.
	 */

	public BookingNote bookTickets(String spectacleName,int[] seats){
		//Controllo che il nome dello spettacolo in input sia nella lista
		for (int i=0;i<spectaclesList.size();i++){
			Spectacle show=(Spectacle)spectaclesList.get(i);
			if (show.getName()==spectacleName){
				//ho trovato lo spettacolo
				int[] bookedSeats=show.getBookedSeats();
				//controllo che i posti da prenotare non lo siano gia'
				for (int j=0;j<seats.length;j++){
					for (int k=0;k<bookedSeats.length;k++){
						if (seats[j]==bookedSeats[k]){
							//i posti sono gia' occupati, prenotazione
annullata
							return null;
						}
					}
				}
				//i posti sono ancora liberi.
				//Concateno i due array
				int[] temp=new int[seats.length+bookedSeats.length];
				for (int j=0;j<seats.length;j++){
					temp[j]=seats[j];
				}
				for (int j=0;j<bookedSeats.length;j++){
					temp[j+seats.length]=bookedSeats[j];
				}
				//aggiorno l'array dei posti prenotati
				show.setBookedSeats(temp);
				//viene aggiornata la lista spettacoli
				spectaclesList.add(i,show);
				//viene creata la ricevuta
				return new BookingNote(spectacleName,seats);
			}
		}
		//non e' stato trovato lo spettacolo nella lista. Prenotazione annullata.
		return null;
	}
```

```
	/**
	 * Servizio per comprare i biglietti, per i quali deve esser stata fatta
	 * una prenotazione con l'apposito servizio bookTickets.
	 * @param note ricevuta di prenotazione
	 * @return true se l'acquisto e' andato a buon fine, altrimenti false.
	 */

	public boolean buyTickets(BookingNote note){
		//Controllo che il nome dello spettacolo nella ricevuta sia nella lista
		for (int i=0;i<spectaclesList.size();i++){
			Spectacle show=(Spectacle)spectaclesList.get(i);
			if (show.getName()==note.getName()){
				//ho trovato lo spettacolo
				int[] seats=note.getSeats();
				int[] boughtSeats=show.getBoughtSeats();
				//controllo che i posti da acquistare non  siano gia' stati acquistati
				for (int j=0;j<seats.length;j++){
					for (int k=0;k<boughtSeats.length;k++){
						if (seats[j]==boughtSeats[k]){
							//i posti sono gia' acquistati, acquisto
annullato
							return false;
						}
					}
				}
				//i posti non sono ancora stati acquistati.
				//Concateno i due array
				int[] temp=new int[seats.length+boughtSeats.length];
				for (int j=0;j<seats.length;j++){
					temp[j]=seats[j];
				}
				for (int j=0;j<boughtSeats.length;j++){
					temp[j+seats.length]=boughtSeats[j];
				}
				//aggiorno l'array dei posti prenotati
				show.setBoughtSeats(temp);
				//viene aggiornata la lista spettacoli
				spectaclesList.add(i,show);
				//l'acquisto e' andato a buon fine
				return true;
			}
		}
		//non e' stato trovato lo spettacolo nella lista. Acquisto annullato.
		return false;
	}

}
```

Listato corrispondente alla classe Service2.class

```
package es04;

import java.io.*;

/**
 * Questa classe rappresenta un prototipo di biglietteria elettronica. Implementa
 * le interfacce TicketService e SpectacleService. Questa classe estende la
 * classe Service modificando i suoi metodi: viene aggiunta la funzione di
 * scrittura su file di testo.
 * @author Andrea Urbini
 */

public class Service2 extends Service{

	/**
	 * Costruisce un oggetto di tipo Service2: una astrazione di biglietteria.
	 */

	public Service2(){
		super();
	}

	/**
	 * Servizio per prenotare i posti di uno spettacolo. Estende il metodo della
	 * classe Service. Se viene creata una prenotazione viene creato un file
	 * service.txt contenente l'informazione sulla prenotazione.
	 * @param spectacleName nome dello spettacolo da ricercare
```

```
         * @param seats array contenente i posti che si intendono prenotare.
         * @return Se l'operazione ha successo viene restituita una prenotazione,
         * al contrario se l'operazione non riesce viene restituito il valore null.
         */

        public BookingNote bookTickets(String spectacleName,int[] seats){
                //Crea la prenotazione richiamendo il metodo della superclasse
                BookingNote note=super.bookTickets(spectacleName,seats);
                //Se la prenotazione fallisce ritorna null
                if (note==null){
                        return null;
                }else{
                        /*Se la prenotazione avviene creo una stringa con l'informazione e
                         *la passo al metodo makeFile.*/
                        String messaggio="Sono stati prenotati "+seats.length+" posti per lo
spettacolo "+spectacleName+". Prenotazone: "+note.getId();
                        makeFile(messaggio);
                        return note;
                }
        }

        /**
         * Servizio per comprare i biglietti, per i quali deve esser stata fatta
         * una prenotazione con l'apposito servizio bookTickets. Se l'acquisto e'
         * fattibile viene creato un file service.txt contenente l'informazione
         * sull'acquisto.
         * @param note ricevuta di prenotazione
         * @return true se l'acquisto e' andato a buon fine, altrimenti false.
         */

        public boolean buyTickets(BookingNote note){
                if (super.buyTickets(note)){
                        /*Se l'acquisto avviene creo una stringa con l'informazione e la
                         *passo al metodo makeFile.*/
                        String messaggio="Sono stati acquistati i biglietti relativi alla
prenotazione: "+note.getId();
                        makeFile(messaggio);
                        return true;
                }
                return false;
        }

        /**
         * Scrive il file service.txt sull'hardDisk contenente le stringhe messaggio.
         * @param messaggio stringa contenente le informazioni da salvare sul file
         */

        private void makeFile(String messaggio){
                FileWriter file=null;
                try{
                        //Crea un oggetto di tipo FileWriter; e' abilitata la funzione append
                        file=new FileWriter("service.txt",true);
                }catch(IOException e1){
                        //Se si verifica una eccezione allora stampo un messaggio di errore
                        System.out.println("Impossibile aprire file");
                        System.exit(1);
                }
                try{
                        //scrive sul file la stringa messaggio
                        file.write("\n"+messaggio+"\n");
                        //Chiude lo stream
                        file.close();
                }catch(IOException e2){
                        //Se si verifica una eccezione allora stampo un messaggio di errore
                        System.out.println("Errore di output");
                        System.exit(2);
                }
        }
}
```

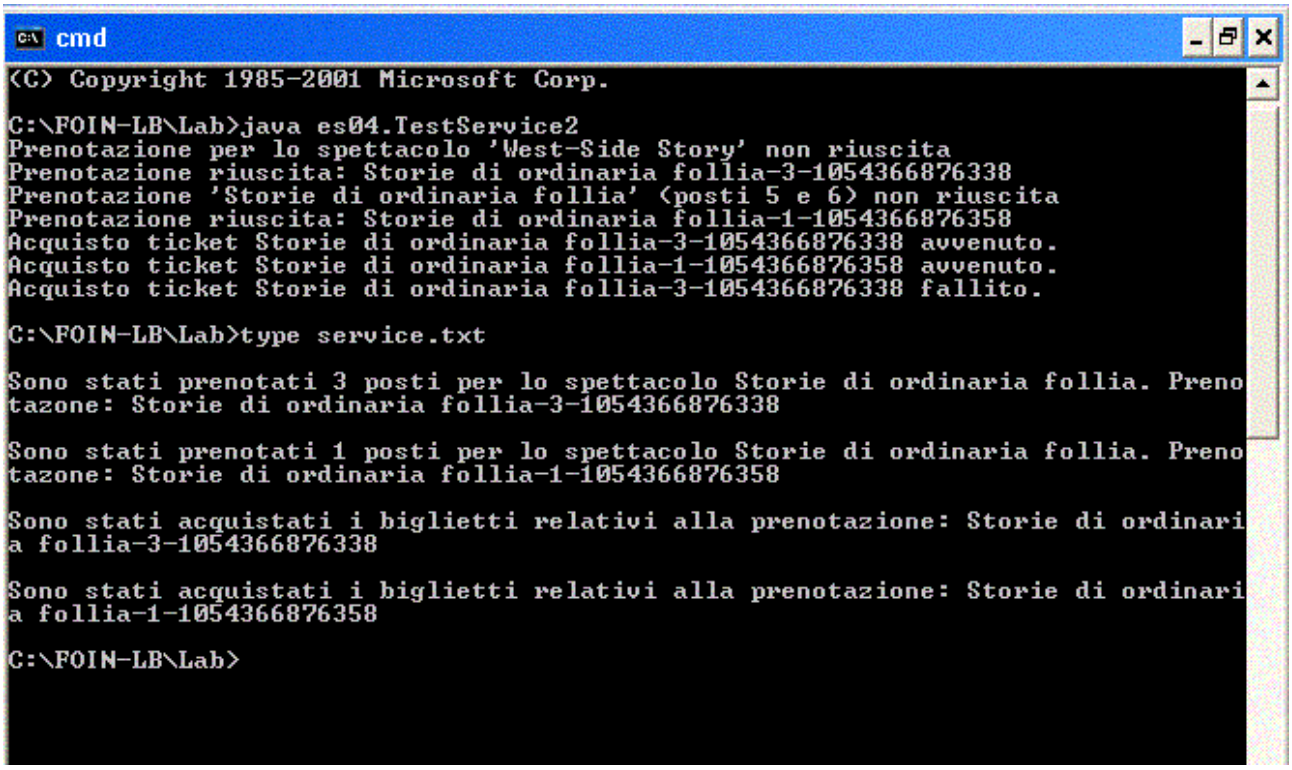
Il listato delle classi TestService.class e TestService2.class è praticamente identico a quello della classe TestTheatre.class fornita dalla ACME, cambiano solo i tipi di oggetti considerati.

Testing e casi d'uso

I componenti software di testing utilizzati sono basati sul modello fornito dalla ACME; corrispondono ai *file TestService.class* e *TestService2.class* che testano, rispettivamente, le classi *Service* e *Service2*. Le fasi di entrambi i test sono le seguenti:

- Creazione di un oggetto *Service* assegnato ai riferimenti *user* di tipo *TicketService* e *owner* di tipo *SpectacleService*. Facendo questa divisione chi utilizza l'oggetto assegnato a *user* ha la limitazione di usare solo i metodi dichiarati nell'interfaccia *TicketService*; mentre all'oggetto assegnato ad *owner* appartiene il solo metodo dell'interfaccia *SpectacleService*.
- Registrazione di spettacoli: si utilizza il metodo *registerSpectacle* applicato all'oggetto assegnato ad *owner*; vengono registrati due spettacoli ("Storie di ordinaria follia" e "Waiting for Godot").
- Prenotazione posti: si utilizza il metodo *bookingSpectacle* e in base al dato restituito si stampa su standard output una stringa. Provando a prenotare i posti per uno spettacolo ("West Side Story") che non è in lista, deve ottenere un errore. Un altro caso di errore si ottiene prenotando posti già prenotati, anche in questo caso il componente software scrive una stringa di errore. Se la prenotazione è avvenuta senza problemi viene stampato su standard output un messaggio contenente i dati della prenotazione.
- Acquisto dei biglietti: si utilizza il metodo *buyTickets* e in base al valore *boolean* restituito si stampa su standard output un messaggio. L'acquisto non avviene se i posti sono già stati acquistati evidentemente con la stessa ricevuta di prenotazione.

Qui sotto riporto la schermata MSDOS ottenuta lanciando la classe *es04.TestService2*, in basso è visibile anche il contenuto del file *service.txt*:



Concetti e tecniche acquisiti

Questa esercitazione mi ha permesso di comprendere le interfacce fino in fondo: dalla spiegazione teorica mi erano sembrate solo uno strumento per classificare e stabilire la signature dei metodi; affrontando questa esercitazione ho notato come le interfacce possano essere utilizzate per limitare i metodi di un oggetto: nel componente software di test si crea un oggetto unico (di tipo *Service*) in cui sono implementati tutti i tre metodi richiesti dalle due interfacce. Quando l'oggetto viene assegnato ai riferimenti *user* (dichiarato *TicketService*) e *owner* (dichiarato *SpectacleService*), l'oggetto "perde" i metodi non dichiarati nell'interfaccia corrispondente al riferimento.
Con la classe *Service2* ho poi messo in pratica quello imparato sul package *java.io* senza problemi.