FONDAMENTI DI INFORMATICA

ESERCIZIARIO JAVA

Ingegneria Gestionale

2017-2018

Presentazione

Per i neofiti della materia, la programmazione, ovvero la scrittura di programmi in un linguaggio di programmazione, può essere tanto ostica quanto di difficile comprensione: la mentalità e la logica con cui un programmatore affronta un problema è, spesso, molto diversa da quella usata nella realtà.
Concetti come classi, metodi, costruttori o variabili possono sembrare oscuri ai più, ma un buon metodo di studio e, soprattutto, una continua pratica sulla materia possono facilitare il superamento delle difficoltà iniziali. E' con quest'intento che nasce il seguente eserciziario: introdurre una materia le cui potenzialità sono facilmente riscontrabili nella vita odierna, in cui la tecnologia la fa da padrona. Soprattutto il linguaggio Java: disponibile gratuitamente sulla Rete, esso permette la creazione di applicazioni eseguibili su un'ampia gamma di computer e di sistemi operativi diversi e, soprattutto, è divenuto lo standard per lo sviluppo delle applicazioni integrate nelle pagine del World Wide Web, o meglio conosciuto come Internet, o di dispositivi quali cellulari o qualsiasi sistema che si basi su un controllo informatico, come i GPS e sensori.
Molti degli esercizi presenti in codesto testo sono stati ricavati dal libro adottato a lezione: "Concetti di informatica e fondamenti di Java" (Cay H. Horstmann, Apogeo Edizioni). Altri, invece, sono esercitazioni proposte in laboratorio dal docente Giorgio Satta (per maggiori informazioni, si rimanda al sito del corso).
L'eserciziario, in ogni caso, non potrà mai sostituire le nozioni espresse in classe o nel libro di testo: esso è una guida che, come Virgilio nella Divina Commedia, accompagna gli studenti all'approfondimento del linguaggio Java.

Struttura:

L'eserciziario è suddiviso in due parti fondamentali:

- Esercizi del libro: risoluzione di parte degli esercizi presenti nel testo ufficiale. Essa è ulteriormente suddivisa in capitoli, all'inizio dei quali si riassumono i concetti fondamentali
- Temi d'esame svolti: esercizi assegnati durante gli esami

Ogni esercizio, inoltre, è strutturato nel seguente modo:

- Testo: consegna dell'esercizio
- Consigli: suggerimenti sulle modalità di sviluppo dei programmi. La loro utilità è puramente soggettiva.

Autori: (in ordine cronologico per anno accademico)

Bressan Marco
Da Giau Alessandro
Dal Maso Alberto
Fioretto Giulio
Gallo Amanda
Gecchele Stefano
Obradovic Tijana
Ronchi Fabrizio
Zulian Marco
Zecchin Giulia
Maculan Marco
Duregon Michele
Parolin Enrico
Rigoni Marco
Zorzi Marco
Pizzato Raissa (2013)
Pizzato Raissa (2017)

Suggerimenti

JCreator e JDK: Il primo è un ambiente di sviluppo per Java, il secondo è la Java Virtual Machine (JVM). Ecco i link da dove è possibile scaricarli:
JCreator: http://www.jcreator.org/download.htm , si consiglia il download nella versione LE (light edition)
JDK: http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html
Documentazione API: è l'insieme di tutte le librerie di Java.
È reperibile al link:
http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html

CAPITOLO 1 Introduzione

REMIND:

(Paragrafo 1.3)

- Java Virtual Machine: → CPU ideale, simulata da un programma in esecuzione sulla CPU effettiva. La JVM perché i codici macchina delle varie CPU (Pentium, Sparc, etc.) hanno linguaggi diversi per le istruzioni macchina. Così facendo tutte le macchine utilizzano lo stesso linguaggio (JVM) → Java può essere codificato indistintamente su qualsiasi macchina.
- Difficile scrivere in linguaggio macchina → codificato come numeri.
- Compilatore → traduce enunciati logici del linguaggio ad alto livello (Java) in sequenze di calcoli del codice macchina.

(Paragrafo 1.4)

- Java pensato per internet, due qualità:
  - sicurezza → caratteristiche di sicurezza per garantire che non si possa scrivere applet[1] nocivi;
  - trasferibilità → Java opera senza bisogno di modifiche su tutti i sistemi operativi (JVM).

(Paragrafo 1.7) Errori

- ERRORE DI SINTASSI → vi è qualcosa di sbagliato secondo le regole del linguaggio e il compilatore lo trova.
- ERRORE DI ESECUZIONE (o LOGICO) → il programma è corretto sintatticamente e fa qualcosa, ma non quello che ci si aspettava.

Esercizio 1.1 – HelloTester

Testo:
Scrivere un programma che stampi sullo schermo il messaggio "Hello, World!".

Consigli:
Concentrarsi, per il momento, sulla riga contenente il messaggio di saluto. Le prime due righe saranno di più facile comprensione con il procedere del corso.

```
public class HelloTester
{
 public static void main(String[] args)
 {
/**Sullo schermo, viene stampato un messaggio di saluti.*/
  System.out.println("Hello, World");
 }
}
```

Esercizio 1.2 – Dave

Testo:
Scrivere un programma che visualizzi sullo schermo del terminale il vostro nome all'interno di un rettangolo, come nell'esempio seguente:

```
+ - - +
| Dave |
+ - - +
```

Consigli:
Stampare il messaggio come composizione di tre righe.

[1] APPLET: Applicazioni java che vengono eseguite all'interno di broswer web. Servono per visualizzare oggetti dinamici che altrimenti con l'html sarebbero difficili da mostrare. Per visualizzarli serve la JVM.

```
public class Dave
{
 public static void main(String[] args)
 {
/**Si stampa la cornice.*/
  System.out.println("+ -  - +");
/**Si stampa il nome.*/
  System.out.println("| Dave |");
/** Si stampa la cornice.*/
  System.out.println("+ -  - +");
 }
}
```

Esercizio 1.3 – LetteraCubitale

Testo:
Scrivere un programma che visualizzi sullo schermo del terminale la vostra iniziale grande e centrata composta da molti caratteri uguali.

```
public class LetteraCubitale
{
 public static void main(String[] args)
 {
/**Si stampano una serie di stringhe affinchè si formi la lettera desiderata.*/
  System.out.println("              FFFFFFFFFFFFF");
  System.out.println("              FFFFFFFFFFFFF");
  System.out.println("              FFF");
  System.out.println("              FFF");
  System.out.println("              FFFFFFFFFF");
  System.out.println("              FFFFFFFFFF");
  System.out.println("              FFF");
  System.out.println("              FFF");
  System.out.println("              FFF");
  System.out.println("              FFF");
 }
}
```

CAPITOLO 2 Utilizzare oggetti

REMIND:

(Paragrafo 2.1) Tipi di variabili:

- A ogni valore è assegnato un tipo (es. String).
- Regole per gli identificatori:
  - possono contenere lettere, cifre, "$", underscore ("_"), ma non possono iniziare con una cifra.
  - No altri simboli (es. "?" o "!").
  - NO SPAZI.
  - Non si possono usare parole riservate.
  - Java è case-sensitivie, quindi distingue maiuscole da minuscole (greeting ≠ Greeting).
- Convenzioni:
  - Le variabili iniziano con la lettera minuscola, le Classi iniziano con la lettera maiuscola.
  - Le variabili, non potendo contenere spazi, utilizzano la "regola del cammello", secondo la quale i nomi vengono concatenati ma ogni nuove parola inizia con la maiuscola (es: unaMiaVariabile).
  - Spesso i metodi contenenti la dicitura "get" (getBase, getAltezza…) restituiscono il valore della dimensione richiesta o eseguono un'operazione specifica (getArea). Quelli contenenti il prefisso "set", invece, permettono di assegnare un nuovo valore.

(Paragrafo 2.2) Operatore di assegnazione

- Si usa "=" (es. int n = 7 → assegna a n il valore 7).

(Paragrafo 2.4) Parametri e valori restituiti dai metodi

- Parametri → dati in ingresso
  Es. System.out.println(greeting)
  greeting = parametro ESPLICITO
  System.out = parametro IMPLICITO → oggetto con cui si invoca il metodo.

- replace → esegue operazioni di ricerca/sostituzione (river.replace("issi", "our"))
- Quando un metodo non restituisce un valore → void
- Nome di un metodo SOVRACCARICO → si riferisce a più di un metodo.
  Es. System.out.println(int n), System.out.println(String s)

(Paragrafo2.7) Metodi di accesso e metodi modificatori

- ACCESSO → restituisce informazioni relative all'oggetto senza modificarlo (es. get()).
- MODIFICATORI → modificano lo stato di un oggetto (es. set()).

(Paragrafo 2.10) Riferimenti a oggetti

- RIFERIMENTO A OGGETTO → la variabile contiene la posizione in memoria di un oggetto: si riferisce a quell'oggetto.
- Le variabili numeriche memorizzano i numeri stessi → RIFERIMENTO A VALORE
- - Quando si copia un valore di tipo primitivo, la copia e l'originale sono indipendenti
  - Quando si copia un riferimento a un oggetto l'originale e la copia sono riferimenti al medesimo oggetto.

(Classe Tester) Testare un programma

- Alla fine di una classe si trova un altro breve programma che di solito viene chiamato NomeDellaClasseTester dove, al posto di NomeDellaClasse, si mette il nome della classe da testare. Infatti i programmi possono contenere errori, una classe tester consiste nell'esecuzione di un frammento di codice per verificare che funzioni correttamente. Un'analisi attenta delle imperfezioni del codice e dei suoi scostamenti da quanto progettato è un passo essenziale per la realizzazione di un programma valido, funzionale, che soddisfi la consegna. In un buon tester sono sempre presenti spiegazioni riguardo ai risultati attesi.

Esercizio 2.1 – Rettangolo

Testo:
Si costruisce un rettangolo partendo da una base, un'altezza e dalle coordinate del piano.

Consigli:
Per chi si affaccia per la prima volta al panorama della programmazione d'informatica, può risultare difficile la risoluzione di tale problema. Si cerchi, quindi, di comprendere ogni passaggio:

- costruttore: ne sono presenti due. Il primo crea un rettangolo con le dimensioni definite, il secondo, invece, permette all'utente di assegnare i valori.

```
public class Rettangolo
{
private int base;
private int altezza;
private int ascissa;
private int ordinata;
```

```
/** Costruttore che crea un rettangolo con dei parametri predefiniti.*/
 public Rettangolo()
 {
  base = 1;
  altezza = 1;
  ascissa = 0;
  ordinata = 0;
 }
/** Costruttore che crea un rettangolo con dei parametri dati.
@param unaBase È la base del rettangolo.
@param unAltezza È l'altezza del rettangolo.
@param unAscissa È l'ascissa del rettangolo.
@param unOrdinata È l'ordinata del rettangolo.*/
 public Rettangolo(int unaBase, int unAltezza, int unAscissa, int unOrdinata)
 {
  base = unaBase;
  altezza = unAltezza;
  ascissa = unAscissa;
  ordinata = unOrdinata;
 }
/** Metodo che restituisce la base del rettangolo.
@return La base del rettangolo.*/
 public int getBase()
 {
  return base;
 }
/** Metodo che restituisce l'altezza del rettangolo.
@return L'altezza del rettangolo.*/
 public int getAltezza()
 {
  return altezza;
 }
/** Metodo che restituisce l'ascissa del rettangolo.
@return L'ascissa del rettangolo.*/
 public int getAscissa()
 {
 return ascissa;
 }
/** Metodo che restituisce l'ordinata del rettangolo.
@return L'ordinata del rettangolo.*/
 public int getOrdinata()
 {
  return ordinata;
 }
/** Metodo che modifica la base del rettangolo.
@param nuovaBase La nuova misura della base.*/
 public void setBase(int nuovaBase)
 {
  base = nuovaBase;
 }
/** Metodo che modifica l'altezza del rettangolo.
@param nuovaAltezza La nuova misura dell'altezza.*/
 public void setAltezza(int nuovaAltezza)
 {
  altezza = nuovaAltezza;
 }
/** Metodo che modifica l'ascissa del rettangolo.
@param nuovaAscissa La nuova ascissa.*/
 public void setAscissa(int nuovaAscissa)
 {
  ascissa = nuovaAscissa;
 }
```

```
/** Metodo che modifica l'ordinata dell'ascissa.
@param nuovaOrdinata La nuova ordinata.*/
 public void setOrdinata(int nuovaOrdinata)
 {
  ordinata = nuovaOrdinata;
 }
/** Metodo che trasla il rettangolo modificandone le coordinate.
@param trX Lo spostamento in ascissa.
@param trY Lo spostamento in ordinata.*/
 public void traslazione(int trX, int trY)
 {
  ascissa = ascissa + trX;
  ordinata = ordinata + trY;
 }
/** Metodo che calcola il perimetro del rettangolo.
@return Il perimetro del rettangolo.*/
 public int getPerimetro()
 {
  return (base + altezza)*2;
 }
/** Metodo che calcola l'area del rettangolo.
@return L'area del rettangolo.*/
 public int getArea()
 {
  return base * altezza;
 }
}



public class RettangoloTester
{
 public static void main(String[] args)
 {
/**Si crea un oggetto di tipo Rettangolo con parametri predefiniti.*/
  Rettangolo r = new Rettangolo();
/**Si stampa il perimetro dell'oggetto Rettangolo.*/
  System.out.println("Perimetro: " + r.getPerimetro());
/**Si stampa l'area dell'oggetto Rettangolo.*/
  System.out.println("Area: " + r.getArea());
  System.out.println("Expected perimetro: 4");
  System.out.println("Expected area: 1");
/**Si crea un altro oggetto di tipo Rettangolo con parametri dati.*/
  Rettangolo rr = new Rettangolo(5, 3, 9, 2);
/**Si stampa il perimetro dell'oggetto Rettangolo.*/
  System.out.println("Perimetro: " + rr.getPerimetro());
/**Si stampa l'area dell'oggetto Rettangolo.*/
  System.out.println("Area: " + rr.getArea());
  System.out.println("Expected perimetro: 16");
  System.out.println("Expected area: 15");
 }
}
```

Esercizio 2.2 – Dado

Testo:

Scrivere un programma che usi la classe Random per simulare il lancio di un dado, visualizzando un numero casuale compreso tra 1 e 6 ogni volta che viene eseguito.

Consigli:

Il seguente programma può essere di difficile comprensione ai neofiti della materia perché viene introdotta la classe Random. Essa, come molte altre classi (Scanner, Rectangle), deve essere importata dalla libreria virtuale di Java,

tramite la dicitura "import nomePacchetto.nomeClasse". Viene anche introdotta la classe Math che va importata nello stesso modo. Si rimanda al ripasso del capitolo 2 l'approfondimento dell'importazione di classi.

```
import java.util.Random;
import java.lang.Math;
public class Dado
{
private int numeroFacce;
private Random gen;
/** Costruttore che crea un dado standard a 6 facce e inizializza
l'oggetto gen.*/
 public Dado()
 {
  numeroFacce = 6;
  gen = new Random();
 }
/** Costruttore che crea un dado con un numero dato di facce e inizializza
l'oggetto gen.
@param numFacce Il numero di facce che avrà il Dado.*/
 public Dado(int numFacce)
 {
  numeroFacce = numFacce;
  gen = new Random();
 }
/** Metodo che cambia il numero di facce del dado.
@param nuovoVal Il nuovo numero di facce del dado.*/
 public void setFacce(int nuovoVal)
 {
  numeroFacce = nuovoVal;
 }
/** Metodo che simula il lancio del dado.
@return Il risultato del lancio.*/
 public int lancia()
 {
  int result = gen.nextInt(numeroFacce) + 1 ;
  return result;
 }
/** Metodo che simula il lancio del dado senza utilizzare l'oggetto di
tipo Random ma utilizzano la classe Math.
@return Il risultato del lancio.*/
 public int lancia2()
 {
  return ((int) (Math.random()* numeroFacce)) + 1;
 }
}
```

```
public class DadoTester
{
 public static void main(String[] args)
 {
/**Si crea un oggetto di tipo Dado standard.*/
  Dado d6 = new Dado();
/**Si crea un oggetto di tipo Dado con parametro predefinito.*/
  Dado d20 = new Dado(20);
/**Si lancia il dado standard(d6) e si stampa il risultato.*/
  int risultato1 = d6.lancia();
  System.out.println("Lancio del d6 usando lancia(): "+ risultato1);
```

```
/**Si lancia il dado con parametro(d20) e si stampa il risultato.*/
  int risultato2 = d20.lancia2();
  System.out.println("Lancio del d20 usando lancia2(): "+ risultato2);
/**Si modifica il parametro.*/
  d20.setFacce(100);
/**Si lancia il dado con parametro modificato e si stampa il risultato.*/
  int risultato3 = d20.lancia2();
  System.out.println("Lancio d20 modificato: " + risultato3);
 }
}
```

CAPITOLO 3 Realizzare classi

REMIND:

(Paragrafo 3.1) Campi di Esemplare

- Un oggetto memorizza i propri dati dentro a *variabili d'istanza* (o *campi di esemplare*).
- VARIABILE D'ISTANZA → zona di memorizzazione presente in ogni oggetto della classe.

(Paragrafo 3.3) Realizzare classi

- Oltre ad appoggiarsi a classe già pronte, si possono creare classi che hanno i propri metodi. Per i dettagli si veda il capitolo 3 del Libro.

(Paragrafo 3.4) Commentare l'interfaccia pubblica

- Spesso conviene commentare i metodi che vengono creati, in modo da rendere più comprensibile il programma ad altri o anche a sé stessi: esistono a tal scopo delle convenzioni:
  - Iniziare il commento, posto prima del metodo in questione, con /** e successivamente con la lettera maiuscola.
  - Terminare ogni frase con un punto.
  - Inserire la dicitura @param seguita dalla spiegazione di cosa è il parametro in questione.
  - Inserire la dicitura @return per spiegare cosa restituisce il metodo.
  - Terminare il commento con */ .

(Paragrafo 3.7) Categorie di variabili

- Categorie:
- Variabili di istanza
- Variabili locali
- Variabili parametro
- DIFFERENZA → tempo di vita e campo di visibilità.

Esercizio 3.1 – BankAccount

Testo:

Es. 3.1: scrivere un programma che costruisca un conto bancario chiamato harrysChecking, versi in esso $1000, prelevi da esso $500, prelevi altri $400 e infine visualizzi il saldo rimanente.
Il programma deve poi creare un altro conto bancario chiamato momsSaving, utilizzando il costruttore che inizializza la variabile balance.
Su quest'ultimo conto deve essere poi applicato un interesse del 10%, a seguito del quale viene stampato il saldo.

Consigli:

Il programma non presenta particolari difficoltà. Si invita a rivedere i consigli dell'esercizio sul rettangolo.

```
public class BankAccount
{
private double balance;
```

```
/** Costruisce un oggetto della classe BankAccount con conto bancario con saldo
uguale a zero.*/
 public BankAccount()
 {
  balance = 0;
 }
/** Costruisce un oggetto della classe BankAccount con conto bancario con un
saldo assegnato.
@param initialBalance Il saldo iniziale. */
 public BankAccount(double initialBalance)
 {
  balance = initialBalance;
 }
/** Metodo che versa denaro nel conto bancario.
@param amount L'importo da versare.*/
 public void deposit(double amount)
 {
  balance = balance + amount;
 }
/** Metodo che preleva denaro dal conto bancario.
@param amount L'importo da prelevare. */
 public void withdraw(double amount)
 {
  balance = balance - amount;
 }
/** Metodo che restituisce il valore del saldo attuale del conto bancario.
@return Il saldo attuale. */
 public double getBalance()
 {
  return balance;
 }
/** Metodo che calcola un interesse sul conto.
@param rate Il tasso d'interesse. */
 public void addInterest(double rate)
 {
  balance = balance + ((balance * rate) / 100);
 }
}

public class BankAccountTester
{
 public static void main(String[] args)
 {
/**Creo un oggetto della classe BankAccount e lo chiamo harrysChecking.*/
  BankAccount harrysChecking = new BankAccount();
/**Deposito sul conto di harrysChecking 1000 dollari.*/
  harrysChecking.deposit(1000);
/**Prelievo dal conto di harrysChecking 500 dollari.*/
  harrysChecking.withdraw(500);
/**Prelievo dal conto di harrysChecking 400 dollari.*/
  harrysChecking.withdraw(400);
/**Stampo il conto di harrysChecking.*/
  System.out.println("Dollars harrysChecking : " +harrysChecking.getBalance());
/**Creo un oggetto della classe BankAccount, con già 1000 dollari in partenza,
 e lo chiamo momsSaving.*/
  BankAccount momsSaving = new BankAccount(1000);
/**Calcolo l'interesse del 10% sul conto momsSavings.*/
  momsSaving.addInterest(10);
/**Stampo il conto di momsSavings.*/
  System.out.println("Dollars momsSaving: " + momsSaving.getBalance());
 }
}
```

Esercizio 3.2 – Cellulare

Testo:

Progettare una classe di nome Cellulare, per rappresentare un telefono cellulare con contratto a ricarica.
Tale classe prevede due variabili d'istanza. La prima variabile d'istanza e' definita come private double carica, e rappresenta il quantitativo di euro disponibile per le chiamate. La seconda variabile d'istanza e' definita come private int numeroChiamate, e rappresenta il numero di chiamate effettuate con il cellulare. La classe deve implementare un costruttore public Cellulare(double unaCarica), che prende come parametro esplicito la quantita' di euro della ricarica iniziale. La classe deve inoltre implementare i seguenti metodi. Un metodo definito come public void ricarica(double unaRicarica), che ricarica il telefonino. Un metodo definito come public void chiama(double minutiDurata), che effettua una chiamata di durata in minuti specificata dal parametro esplicito. Tale metodo dovra' aggiornare la carica disponibile, ed incrementare la memoria contenente il numero di chiamate effettuate dal telefonino. Si assuma un costo di 0.20 euro per ogni minuto di chiamata. Un metodo public double numero404(), che restituisce il valore della carica disponibile. Un metodo public int getNumeroChiamate(), che restituisce il valore della variabile d'istanza numeroChiamate. Infine, un metodo public void azzeraChiamate(), che azzera la variabile contenente il numero di chiamate effettuate dal telefonino.

Consigli:

S'invita a porre particolare attenzione al metodo chiamata(double minutiDurata), che aggiorna il credito e incrementa di una unità il numeroChiamate.

```
public class Cellulare
{
private double credito;
private int numeroChiamate;
/**Creo un oggetto della classe Cellulare.
@param creditoIniziale Credito iniziale.
@param nChiamateIniziali Numero di chiamate iniziali.*/
 public Cellulare(double creditoIniziale, int nChiamateIniziali)
 {
  credito = creditoIniziale;
  numeroChiamate = nChiamateIniziali;
 }
/**Ricarica il cellulare.
@param unaRicarica Importo della ricarica.*/
 public void ricarica(double unaRicarica)
 {
  credito = credito + unaRicarica;
 }
/**Effettua una chimata, aggiorna il credito, incrementa di una unità il
numeroChiamate.
@param minutiDurata Durata della chiamata.*/
 public void chiamata(double minutiDurata)
 {
  credito = credito - (0.20*minutiDurata);
  numeroChiamate = numeroChiamate + 1;
 }
/**Restituisce il valore del credito.
@return Credito.*/
 public double numero404()
 {
  return credito;
 }
```

```
/**Restituisce il numero di chiamate.
@return Numero di chiamate.*/
 public int getNumeroChiamate()
 {
  return numeroChiamate;
 }
/**Azzera il numero di chiamate.*/
 public void azzeraChiamate()
 {
  numeroChiamate = 0;
 }
}
```

```
public class CellulareTester
{
 public static void main(String[] args)
 {
/**Creo un oggetto della classe Cellulare,
con 20€ di credito e 0 chiamate effettuate.*/
  Cellulare nokia = new Cellulare(20.0,0);
/**Ricarico il cellure di 50€.*/
  nokia.ricarica(50.0);
/**Effettuo una chiamata di 13 minuti.*/
  nokia.chiamata(13.0);
/**Stampo il credito.*/
  System.out.println(nokia.numero404());
/**Stampo l'aspettativa di credito.*/
  System.out.println("Credito, mi aspetto: 67.4");
/**Stampo il numero di chiamate effettuate.*/
  System.out.println(nokia.getNumeroChiamate());
/**Stampo l'aspettativa del numero di chiamate effettuate.*/
  System.out.println("Numero Chiamate, mi aspetto: 1");
/**Azzero il numero di chiamate effettuate.*/
  nokia.azzeraChiamate();
/**Stampo il numero di chiamate effettuate.*/
  System.out.println(nokia.getNumeroChiamate());
/**Stampo l'aspettativa del numero di chiamate effettuate.*/
  System.out.println("Numero Chiamate, mi aspetto: 0");
 }
}
```

Esercizio 3.3 – Car

Testo:

Progettare e realizzare una classe Car(automobile) con le proprietà seguenti. Un'automobile ha una determinata resa del carburante (misurata in miglia/galloni o in litri/chilometri: scegliete il sistema che preferite) e una certa quantità di carburante nel serbatoio. La resa è specificata dal costruttore e il livello iniziale del carburante è a zero. Fornire questi metodi: un metodo drive per simulare il percorso di un'automobile per una certa distanza, riducendo il livello di carburante nel serbatoio; un metodo getGas, per ispezionare il livello corrente del carburante; un metodo addGas per far rifornimento.

Consigli:

S'invita a porre particolare attenzione al metodo drive(double km), che calcola il livello di carburante dopo un certo percorso.

```
public class Car
{
private double gas;
private double resa;

/** Costruttore che costruisce un'automobile con carburante uguale a zero.*/
 public Car(double kmL)
 {
  resa = kmL;
  gas = 0;
 }
/** Metodo che calcola il livello di carburante rimasto dopo un certo percorso.
@param km I chilometri percorsi.*/
 public void drive(double km)
 {
  gas = gas - (km / resa);
 }
/** Metodo che restituisce la quantità di carburante rimasto.
@return Il carburante rimasto.*/
 public double getGas()
 {
  return gas;
 }
/** Metodo che aggiunge carburante nel serbatoio.
@param rifornimento Il carburante da aggiungere.*/
 public void addGas(double rifornimento)
 {
  gas = gas + rifornimento;
 }
}


public class CarTester
{
 public static void main(String[] args)
 {
/**Creo un oggetto della classe Car.*/
  Car ibiza = new Car(20);
/**Aggiungo 20 litri di carburante nel serbatoio di ibiza.*/
  ibiza.addGas(20);
/**Faccio percorrere 100 Km a ibiza.*/
  ibiza.drive(100);
/**Stampo la quantità di carburante rimasto.*/
  System.out.println("Quantità di carbunate:"+ibiza.getGas());
/**Stampo l'aspettativa della quantià di carburante nel serbatoio.*/
  System.out.println("Quantità di carbunate, mi aspetto:15.0");
 }
}
```

Esercizio 3.4 – DistributoreBenzina

Testo:

Progettare una classe di nome DistributoreBenzina, per rappresentare un distributore di carburante per automobili. Tale classe prevede due variabili d'istanza. La prima chiamata deposito, di tipo double, e contiene il quantitativo di benzina disponibile al distributore. La seconda, chiamata euroPerLitro, di tipo double, rappresenta il prezzo della benzina, espresso in euro per litro.

La classe deve implementare un costruttore public DistributoreBenzina(double unPrezzoPerLitro), che prende come parametro esplicito il prezzo in euro per un litro di benzina. La quantita' iniziale di benzina disponibile e' zero. La

classe deve inoltre implementare i seguenti metodi: public void rifornisci(double unaQuantita), che rifornisce il distributore di benzina. public void vendi(double euro, Car unaAutomobile) che vende una quantita' di benzina corrispondente all'ammontare di euro pagato che va a rifornisce l'automobile passata come parametro esplicito. public void aggiorna(double unPrezzoPerLitro), che aggiorna il prezzo della benzina.

Consigli:
Si noti che nel metodo vendi(double euro, Car unAutomobile) il secondo parametro esplicito unAutomobile è un oggetto della classe Car dell'esercizio 3.3 che deve essere costruito con il costruttore della classe Car, di conseguenza, si possono sfruttare i metodi della classe Car.
Nel Tester create varie automobili e un distributore. Fate compiere alcuni "viaggi" alle automobili, e rifornitele di benzina in modo appropriato. Inoltre, rifornite la pompa di benzina in caso di necessità.

```
public class DistributoreBenzina
{
private double deposito;
private double euroPerLitro;
/**Costruttore che crea un distributore vuoto con un prezzo assegnato.
@param unPrezzoPerLitro sono gli euro/litro del carburante.*/
 public DistributoreBenzina(double unPrezzoPerLitro)
 {
  euroPerLitro = unPrezzoPerLitro;
  deposito = 0;
 }


/**Metodo che rifornisci di benzina il deposito.
@param unaQuantita quantità rifornita.*/
 public void rifornisci(double unaQuantita)
 {
  deposito = deposito + unaQuantita;
 }
/**Metodo che vende carburante togliendone dal deposito,
 e aggiungendolo ad un oggetto della classe Car.
@param euro Quantitavi in euro venduto.
@param unAutomobile Oggetto della classe Car.*/
 public void vendi(double euro, Car unAutomobile)
 {
  double litriVenduti = euro/euroPerLitro;
  deposito = deposito - litriVenduti;
  unAutomobile.addGas(litriVenduti);
 }

/**Metodo che aggiorna il prezzo della benzina al litro.
@param unPrezzoPerLitro È il nuovo prezzo di un litro.*/
 public void aggiorna(double unPrezzoPerLitro)
 {
  euroPerLitro = unPrezzoPerLitro;
 }
/**Metodo che restituisce la quantità di benzina nel deposito.
@return La quantità di carburante nel deposito.*/
 public double getBenzina()
 {
  return deposito;
 }
}
```

```
public class DistributoreBenzinaTester
{
 public static void main (String[] args)
 {
/**Creo un oggetto della classe DistribitoreBenzina2,
 lo chiamo esso, imposto 1.539 euro al litro.*/
  DistributoreBenzina esso = new DistributoreBenzina(1.539);
/**Rifornisco il deposito di 100 litri.*/
  esso.rifornisci(100);
/**Stampo la quantità di benzia presente nel deposito.*/
  System.out.println(esso.getBenzina());
/**Creo un oggetto della classe Car2,
 lo chiamo punto, gli imposto una resa di 15 Km/l.*/
  Car punto = new Car(15);
/**Rifornisco la punto di 20 euro di benzina.*/
  esso.vendi(20, punto);
/**Stampo la quantità di benzia presente nel deposito.*/
  System.out.println(esso.getBenzina());
/**Stampo la quantità di benzia presente nel serbatoio della punto.*/
  System.out.println(punto.getGas());
/**Faccio percorrere alla punto 20 Km.*/
  punto.drive(20);
/**Stampo la quantità di benzia presente nel serbatoio della punto.*/
  System.out.println(punto.getGas());
/**Aggiorno il prezzo della benzina.*/
  esso.aggiorna(1.542);
 }
}
```

Esercizio 3.5 – Dipendente

Testo:

Progettare una classe di nome Dipendente. Tale classe prevede due variabili di istanza una definita come nome, di tipo String e una definita come stipendio di tipo double. Scrivere un costruttore senza parametri, un costruttore con due parametri (nome e stipendio).La classe deve inoltre implementare i seguenti metodi. Un metodo definito come public String getNome() che restituisce il nome del dipendente. Un metodo definito come public double getStipendio() che restituisce lo stipendio del dipendente. Un metodo public void setStipendio(double nuovoStipendio) che modifica il valore dello stipendio. Un metodo public void setNome(String nuovoNome) che cambia il nome al dipendente. Aggiungere poi un metodo aumento(double percentuale), che incrementi lo stipendio del dipendente secondo una certa percentuale.

Consigli:

Il seguente esercizio è simile a quelli precedenti. Lo scopo risulta semplicemente acquisire una metodologia di lavoro.
Da notare che inizialmente al nome viene fatto assumere il valore “”: questo significa che la stringa è vuota, perché il costruttore non ha paramentri quindi non ci sono informazioni riguardo al nome del dipendente.

```
public class Dipendente
{
private String nome;
private double stipendio;
/**Costruttore senza parametri che costruisce un oggetto della classe
Dipendente.*/
```

```
 public Dipendente()
 {
  nome = "";
  stipendio = 0;
 }
/**Costruttore che inizializza le variabili con valori predefiniti.
@param unNome Il nome del del dipendente.
@param unoStipendio Lo stipendio del dipendente.*/
 public Dipendente(String unNome, double unoStipendio)
 {
  nome = unNome;
  stipendio = unoStipendio;
 }
/**Metodo che restituisce il nome del dipendente.
@return Il nome del del dipendente.*/
 public String getNome()
 {
  return nome;
 }
/**Metodo che restituisce lo stipendio del dipendente.
@return Lo stipendio.*/
 public double getStipendio()
 {
  return stipendio;
 }
/**Metodo che modifica il valore dello stipendio.
@param nuovoStipendio Il nuovo valore dello stipendio.*/
 public void setStipendio(double nuovoStipendio)
 {
  stipendio = nuovoStipendio;
 }

/**Metodo che modifica la stringa del nome.
@param nuovoNome Il nuovo nome dell'operaio.*/
 public void setNome(String nuovoNome)
 {
  nome = nuovoNome;
 }

/**Metodo che incrementa lo stipendio del dipendente.
@param percentuale Il tasso di aumento dello stipendio.*/
 public void aumento(double percentuale)
 {
  stipendio = stipendio + ((stipendio * percentuale) / 100);
 }
}



public class DipendenteTester
{
 public static void main(String[] args)
 {
/**Creo un oggetto di classe Dipendente, lo chiamo d.*/
  Dipendente d = new Dipendente();
/**Imposto un nome a d.*/
  d.setNome("Andrea");
/**Inserisco il valore dello stipendio.*/
  d.setStipendio(1000.0);
/**Incremento del 10% lo stipendio.*/
  d.aumento(10.0);
/**Stampo il nome e lo stipendio del dipendente.*/
  System.out.println(d.getNome() + " = " + d.getStipendio());
```

```
/**Creo un altro oggetto di classe Dipendente, lo chiamo a.*/
  Dipendente a = new Dipendente("Mario", 1500);
/**Incremento del 10% lo stipendio.*/
  a.aumento(10.0);
/**Stampo il nome e lo stipendio del dipendente.*/
  System.out.println(a.getNome() + " = " + a.getStipendio());
 }
}
```

CAPITOLO 4 Tipi di dati fondamentali

REMIND:

(Paragrafo 4.1) Tipi di numeri

- *BigInteger* → classe con rispettivi metodi: non è un tipo primitivo, si usa con numeri molto grandi.
- *BigDecimal* → come BigInteger. Consente di effettuare calcoli in virgola mobile senza errori di approssimazione.
- Con valori in virgola mobile bisogna tenere presente il problema degli errori di arrotondamento.
- (int) → cast, converte in numero intero ignorandone la parte frazionaria (es. 13.75 → 13)
- Usate long per arrotondare un numero in virgola mobile: se il valore assoluto è elevato int potrebbe non essere sufficiente:
  long n = Math.round(d); (13.75 → 14)

(Paragrafo 4.2) Costanti

- final → indica le costanti in Java.
- Per convenzione si usano le lettere maiuscole (es. MAX_VALUE).
- Se si usano in più metodi si devono dichiarare insieme alle variabili d'istanza.
- Non c'è pericolo nel dichiararle come "public" perché non possono essere modificate.

(Paragrafo 4.4) Aritmetica e funzioni matematiche

- "/" → divisione (es. 15 / 4).
- Se si dividono numeri interi, il risultato sarà un intero(es. 15 / 4 = 3).
- "%" (percentuale) → resto (es. 15 % 4 = 3).

(Paragrafo 4.5) Invocare metodi statici

- Vengono richiamati con il nome della classe a cui appartengono, senza l'ausilio di alcun oggetto.
- Sono sempre definiti all'interno di classi → specificare la classe a cui appartiene il metodo (es. Math.sqrt(x);).

(Paragrafo 4.6) Stringhe

- Studia in questo paragrafo i vari metodi per la classe String
- Con l'operatore "+" si possono concatenare stringhe
- Integer.parseInt(s) → converte la stringa s in un numero intero (esiste anche parseDouble())
- Integer.toString(n) → converte il numero n in una stringa

(Paragrafo 4.7) Leggere dati in ingresso

- Scanner → consente la lettura dei dati inseriti in ingresso dalla tastiera
- Studia in questo paragrafo i vari metodi per la classe Scanner
- ATTENZIONE: PRECISAZIONE SULLA LETTURA DATI IN INGRESSO

nextInt(), next(), nextDouble() e compagnia leggono tutto quello che devono leggere e si fermano al primo carattere separatore incontrato (bianco, tab, a capo) lasciando tale carattere nel canale di ingresso (cioe` non viene prelevato). Per contro, nextLine() prende tutta la riga di caratteri sino al prossimo carattere a capo compreso.
Allora se digitiamo ad es. 231 seguito da a capo, nextInt() preleva 231 e lascia a capo nel canale. Quanto arriva nextLine() legge una riga vuota, rimuove a capo e si ferma subito. Quindi non bisognerebbe mescolare i due tipi di istruzioni di input, a meno che non si sappia bene quello che si sta facendo.

Esercizio 4.1 – PowerGenerator

Testo:

Scrivere un programma che stampi i seguenti valori:

```
1
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
10000000000
100000000000
```

Consigli:

Si rammenta, come già enunciato negli esercizi del capitolo 2, la necessità di importare una libreria Java per poter eseguire talune operazioni: in questo caso, sarà la classe Math. Nel caso specifico, essa sarà utile per creare l'elevamento a potenza. Si ricorda, inoltre, di incrementare la variabile d'istanza "exp". Da questo esercizio in poi i Tester saranno privi di commenti.

```
import java.lang.Math;
public class PowerGenerator
{
private double base;
private int exp;
/**Creo un oggetto della classe PowerGenerator.*/
 public PowerGenerator()
 {
  base = 0.0;
  exp = 0;
 }
/**Metodo che imposta il valore della base.
@param b Il valore da impostare.*/
 public void setBase(double b)
 {
  base = b;
 }
/**Metodo che calcola la potenza del numero,
l'esponente viene incrementato di uno ogni volta che viene
richiamato il seguente metodo.
@return Il numero elevato a exp.*/
 public double nextPow()
 {
  double result = Math.pow(base, exp);
  exp = exp + 1;
  return result;
 }
}
```

```
public class PowerGeneratorTester
{
 public static void main(String[] args)
 {
  PowerGenerator p = new PowerGenerator();
  p.setBase(10);
  System.out.println(p.nextPow());
  System.out.println(p.nextPow());
  System.out.println(p.nextPow());
 }
}
```

Esercizio 4.2 – TimeConverter

Testo:

Scrivere un programma che, dato un certo tempo in giorni, ore, minuti e secondi, restituisca il numero totale di secondi.

Consigli:

Si prega di osservare che un giorno ha 86400 secondi, un'ora ha 3600 secondi e un minuto 60 secondi. Per la prima volta viene utilizzata la classe Scanner (è necessario importarla) per la stesura del Tester, si prega di leggere gli ultimi punti del Remind capitolo 4.

```
public class TimeConverter
{
public static final int GIORNOSEC = 86400;
public static final int ORESEC = 3600;
public static final int MINUTISEC = 60;
private int s, m, h, d;
/** Costruttore che costruisce un covertitore da giorni, ore, minuti,
secondi, in secondi totali.
@param dd Giorni.
@param hours Ore.
@param min Minuti.
@param sec Secondi.*/

 public TimeConverter(int dd, int hours, int min, int sec)
 {
  d = dd;
  h = hours;
  m = min;
  s = sec;
 }
/**Metodo che restituisce i secondi totali a partire da gg, h, min, sec
inseriti.
@return I secondi totali.*/
 public int getSecTot()
 {
  int sTot = d* GIORNOSEC + h* ORESEC + m* MINUTISEC + s;
  return sTot;
 }
}
```

```
import java.util.Scanner;
public class TimeConverterTester
{
 public static void main(String[] args)
 {
```

```
  Scanner in = new Scanner(System.in);
  System.out.println("Inserisci il numero dei giorni ");
  int dd = in.nextInt();
  System.out.println("Inserisci il numero delle ore ");
  int hours = in.nextInt();
  System.out.println("Inserisci il numero dei minuti ");
  int min = in.nextInt();
  System.out.println("Inserisci il numero dei secondi ");
  int sec = in.nextInt();
  TimeConverter conv = new TimeConverter(dd, hours, min, sec);
  System.out.println("Secondi totali : " + conv.getSecTot());
 }
}
```

Esercizio 4.3 – SecondConverter

Testo:

Scrivere un programma che, dato un certo numero di secondi restituisca il numero di giorni, ore, minuti e secondi.

Consigli:

Si preferisce implementare quattro metodi separati per ottenere il numero di giorni, ore, minuti o secondi. Le uniche difficoltà riscontrabili nell'esercizio riguardano la serie di calcoli da dover svolgere per rendere efficace la conversione. Si prega di osservare che un giorno ha 86400 secondi, un'ora ha 3600 secondi e un minuto 60 secondi.

```
public class SecondConverter
{
private int sTot;
private final int SECONDI_AL_MINUTO = 60;
private final int SECONDI_ALL_ORA = 3600;
private final int SECONDI_AL_GIORNO = 86400;
/**Costruttore costruisce un convertitore da secondi a gg, h, min, sec.
@param secondi I secondi totali.*/
 public SecondConverter(int secondi)
 {
  sTot = secondi;
 }
/**Metodo che restituisce il numero dei giorni.
@return dLocal Giorni.*/
 public int getDays()
 {
  int dLocal = sTot / SECONDI_AL_GIORNO;
  return dLocal;
 }
/**Metodo che restituisce il numero delle ore.
@return hLocal Ore.*/
 public int getHours()
 {
  int sLocal = sTot % SECONDI_AL_GIORNO;
  int hLocal = sLocal / SECONDI_ALL_ORA;
  return hLocal;
 }
/**Metodo che restituisce il numero dei minuti.
@return mLocal Minuti.*/
 public int getMin()
 {
  int sLocal = (sTot % SECONDI_AL_GIORNO)% SECONDI_ALL_ORA;
  int mLocal = sLocal / SECONDI_AL_MINUTO;
  return mLocal;
 }
/** Metodo che restituisce il numero dei secondi.
@return sLocal Secondi.*/
```

```
 public int getSec()
 {
  int sLocal = ((sTot % SECONDI_AL_GIORNO)% SECONDI_ALL_ORA)%SECONDI_AL_MINUTO;
  return sLocal;
 }
}
```

```
import java.util.Scanner;
public class SecondConverterTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Iserisci il numero dei secondi totali ");
  int secTot = in.nextInt();
  SecondConverter conv = new SecondConverter(secTot);
  System.out.println("Giorni: " + conv.getDays());
  System.out.println("Ore: " + conv.getHours());
  System.out.println("Minuti: " + conv.getMin());
  System.out.println("Secondi: " + conv.getSec());
 }
}
```

CAPITOLO 5 Decisioni

REMIND:

(Paragrafi 5.1 - 5.2 - 5.3) Enunciato if e suoi utilizzi

- L'enunciato if permette di scegliere tra diverse alternative.
- Si costruisce così: *if (condizione) istruzione;*
- Si possono anche inserire più di un istruzone, racchiudendole tra due graffe.
- Per fare dei confronti tra valori nelle condizioni si usano i seguenti operatori: <, >, <=, >=, ==, !=.
- Si possono anche costruire alternative multiple, in questo modo:

```
if (condizione) istruzione;
else if (condizione 2) istruzione 2;
else istruzione 3;
```

(Paragrafo 5.4) Utilizzare operatori booleani

- && → and
- || → or
- ! → not

(Metodi ausiliari) Utilizzo di metodi ausiliari

- Sono metodi privati, usati solo dentro la classe, per facilitare l'implementazione dei metodi pubblici richiesti.

Esercizio 5.1 – InputChecker

Testo:

Scrivere un programma che stampi la domanda "Vuoi continuare?" e che attenda dati dall'utente. Se l'utente immette "S", "Sì", "Ok", "Certo" o "Perché no?",

stampare "OK". Se l'utente scrive "N" o "No", stampare "Fine". Negli altri casi, stampare "Dato non corretto". Non considerare differenze tra maiuscolo e minuscolo, quindi anche "s" e "sì" sono validi.

Consigli:
Per la prima volta, l'esercizio introduce l'espressione decisionale "if", cioè definisce il comportamento di un programma, se e solo se sono rispettate determinate condizioni. Nel metodo String risposta(), è consigliabile concatenare tutte le condizioni richieste dall'esercizio.

```
public class InputChecker
{
private String answer;
/**Costruttore che crea un confrontatore di risposte.
@param aAnswer La risposta da confrontare.*/
 public InputChecker(String aAnswer)
 {
  answer = aAnswer;
 }
/**Metodo che restituisce il risultato della risposta.
@return Il risultato della risposta.*/
 public String risposta()
 {
  if (answer.equalsIgnoreCase("S")||answer.equalsIgnoreCase("Sì")||
      answer.equalsIgnoreCase("Certo")||answer.equalsIgnoreCase("OK")||
      answer.equalsIgnoreCase("Perché no?"))
    return "OK";
  else if(answer.equalsIgnoreCase("N")|| answer.equalsIgnoreCase("No"))
    return "Fine";
  else
    return"Dato non corretto!";
 }
}
import java.util.Scanner;
public class InputCheckerTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Vuoi continuare ? (s/n)");
  String risp = in.nextLine();
  InputChecker input = new InputChecker(risp);
  System.out.println(input.risposta());
 }
}
```

Esercizio 5.2 – Sort

Testo:
Scrivere un programma che riceva tre numeri in virgola mobile come dati in ingresso, per poi stamparli in ordine crescente.

Consigli:
L'esercizio non presenta difficoltà degne di nota. Si consiglia, però, di fare un piccolo schema di tutti i casi possibili richiesti dall'esercizio, in modo da evitare errori logici. Si ricorda di inserire i tre valori con la virgola(,) e non con il punto(.).

```
public class Sort
{
private double a, b, c;
/**Creo un oggetto della classe Sort.*/
```

```
 public Sort()
 {
  a = 0.0;
  b = 0.0;
  c = 0.0;
 }
/**Metodo che ordina i numeri.
@param n,m,d i numeri inseriti*/
 public void sortNum(double n, double m, double d)
 {
  double temp = 0.0;
  a = n;
  b = m;
  c = d;
  if ((a < b)&&(c < b))
    if (c < a)
    {
     temp = a;
     a = c;
     c = b;
     b = temp;
    }
    else
    {
     temp = b;
     b = c;
     c = temp;
    }
  else if (b < a)
    if(c < a)
      if(b < c)
      {
       temp = a;
       a = b;
       b = c;
       c = temp;
      }
      else

      {
       temp = a;
       a = c;
       c = temp;
      }
    else
    {
     temp = a;
     a = b;
     b = temp;
    }
 }
/**Metodo che restituisce a.
@return a Il valore di a.*/
 public double getA()
 {
  return a;
 }
/**Metodo che restituisce b.
@return d Il valore di b.*/
 public double getB()
 {
  return b;
 }
```

```
/** Metodo che restituisce c.
@return c Il valore di c.*/
 public double getC()
 {
  return c;
 }
}


import java.util.Scanner;
public class SortTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Inserire tre nemeri: ");
  double a = in.nextDouble();
  double b = in.nextDouble();
  double c = in.nextDouble();
  Sort s = new Sort();
  s.sortNum(a, b, c);
  System.out.println("I valori inseriti in ordine crescente: ");
  System.out.println(s.getA());
  System.out.println(s.getB());
  System.out.println(s.getC());
 }
}
```

Esercizio 5.3 – Year

Testo:

Un'anno con 366 giorni è detto anno bisestile. Un anno è bisestile se è divisibile per quattro (per esempio, il 1980), ma, dopo l'adozione del calendario gregoriano avvenuta il 15 ottobre 1582, un anno non è bisestile se è divisibile per 100 (per esempio, il 1900), mentre è bisestile se è divisibile per 400 (per esempio, il 2000). Scrivete un programma che chieda all'utente di inserire un anno e che calcoli se l'anno è bisestile. Scrivere una classe Year con un metodo predicativo boolean isLeapYear().

Consigli:

Come nell'esercizio precedente, si invita a controllare attentamente le condizioni per cui un anno è bisestile.

```
public class Year
{
private int anno;
/**Crea un oggetto della classe Year.
 @param unAnno Un anno.*/
 public Year(int unAnno)
 {
  anno = unAnno;
 }
/**Un anno è bisestile se è divisibile per 400 oppure per 4 ma non per
100. Questo metodo controlla tale proprietà.
@return True se è bisestile, false se non lo è.*/
 public boolean isLeapYear()
 {
  int resto1 = anno % 400;
  int resto2 = anno % 4;
  int resto3 = anno % 100;
  return ((resto2 == 0 && resto3 != 0) || resto1 == 0);
 }
```

```
/**Metodo che restituisce il valore della variabile anno.
@return anno L'anno.*/
 public int getYear()
 {
  return anno;
 }
}


import java.util.Scanner;
public class YearTester
{
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);
  System.out.println("Inserisci un anno: ");
  Year anno = new Year(input.nextInt());
  if(anno.isLeapYear())
    System.out.println("L'anno "+anno.getYear()+" è bisestile.");
  else
    System.out.println("L'anno "+anno.getYear()+" non è bisestile.");
 }
}
```

Esercizio 5.4 – Voti

Testo:

Scrivete un programma per convertire la lettera di un voto scolastico nel numero corrispondente. Le lettere sono A, B, C, D e F, eventualmente seguite dai segni + o -. I loro valori numerici sono 4, 3, 2, 1 e 0. F+ e F- non esistono. Un segno + o – incrementa o decrementa il valore numerico di 0.3. Tuttavia, A+ è uguale a 4.0. Usate una classe Grade con un metodo getNumericGrade.

Consigli:

Si consiglia di intendere il voto come un double, anziché una stringa. Ciò semplifica l'esecuzione del programma: in tal modo, il voto può essere modificato tramite semplici operazioni aritmetiche.

```
public class Voti
{
private String votoL;
private double votoN;
private String segno;
/**Per il segno + si aggiunge,per il segno - si sottrae.*/
final static double SEGNO = 0.3;
/**Creo un oggetto della classe Voti.*/
 public Voti()
 {
  votoL = "";
  votoN = 0;
  segno = "";
 }
/**Metodo che restituisce un numero a partire dal voto in lettere.
@param input È il voto in lettere.
@return Il voto espresso in numero.*/
 public double getNumericGrade(String input)
 {
  votoL = input.substring(0,1);
  segno = input.substring(1);
  if (votoL.equalsIgnoreCase("F"))
    votoN = 0;
  else
  {
```

```
   if (votoL.equalsIgnoreCase("A"))
     votoN = 4;
   else if (votoL.equalsIgnoreCase("B"))
     votoN = 3;
   else if (votoL.equalsIgnoreCase("C"))
     votoN = 2;
   else if (votoL.equalsIgnoreCase("D"))
     votoN = 1;
   if ((segno.equals("+"))&&(!votoL.equalsIgnoreCase("A")))
     votoN = votoN + SEGNO;
   else if (segno.equals("-"))
     votoN = votoN - SEGNO;
  }
  return votoN;
 }
}
```

```
import java.util.Scanner;
public class VotiTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  Voti voto1 = new Voti();
  boolean c = true;
  String input;
  System.out.println("Scrivi il voto da A a F da convertire.Premi Q per
uscire");
  input = in.next();
  if (input.equalsIgnoreCase("Q"))
    c = false;
  else
    System.out.println(voto1.getNumericGrade(input));
 }
}
```

Esercizio 5.5 – NumberConverter

Testo:

Si scriva un metodo arabicToString che converta un numero nell'intervallo [1, 999.999] in una stringa con il nome del numero in italiano. Ad es 247.247 viene convertito nella stringa duecentoquarantasettemiladuecentoquarantasette.

Consigli:

Si scomponga il problema in casi più semplici considerando la simmetria e la ripetitività di alcuni eventi. Si devono creare nuovi oggetti della classe NumberConverter da cui invocare i metodi per convertire le centinaia, le decine e le unità.

```
public class NumberConverter
{
private int number = 0;
/**Crea un oggetto della classe NumberConverter.
 @param aNumber Un numero.*/
 public NumberConverter(int aNumber)
 {
  number = aNumber;
 }
/**Metodo che converte un numero compreso tra 0 e 999.999 dalla forma in cifre a
quella in lettere.
@return Il numero convertito.*/
```

```
public String arabicToString()
 {
  String result = "";
  if (number == 0)
    result= "zero";
  else
  {
   int sx = number / 1000;
   int dx = number % 1000;
   NumberConverter s = new NumberConverter(sx);
   NumberConverter d = new NumberConverter(dx);
   if (sx == 1)
     result = "mille";
   else if (sx > 1)
     result = s.centinaiaToString() + "mila";
   result = result + d.centinaiaToString();
  }
  return result;
 }
/**Metodo che converte un numero compreso tra 0 e 999 dalla forma in cifre a
quella in caratteri.
@return Il numero convertito.*/
 public String centinaiaToString()
 {
  String result = "";
  int sx = number / 100;
  int dx = number % 100;
  NumberConverter s = new NumberConverter(sx);
  NumberConverter d = new NumberConverter(dx);
  if (sx > 1)
    result = s.unitàToString() + "cento";
  else if (sx == 1)
    result = "cento";
  result = result + d.decineToString();
  return result;}
/**Metodo che converte un numero compreso tra 0 e 99 dalla forma in cifre a
quella in caratteri.
@return Il numero convertito.*/
 public String decineToString()
 {
  String result ="";
  int sx = number / 10;
  int dx = number % 10;
  NumberConverter d = new NumberConverter(dx);
  if(sx == 9)
    result = "novanta" + d.unitàToString();
  else if(sx == 8)
    result = "ottanta" + d.unitàToString();
  else if(sx == 7)
    result = "settanta" + d.unitàToString();
  else if(sx == 6)
    result = "sessanta" + d.unitàToString();
  else if(sx == 5)
    result = "cinquanta" + d.unitàToString();
  else if(sx == 4)
    result = "quaranta" + d.unitàToString();
  else if(sx == 3)
    result = "trenta" + d.unitàToString();
  else if(sx == 2)
    result = "venti" + d.unitàToString();
  else if(sx == 0)
    result = d.unitàToString();
```

```
  else
  {
   if(dx == 0)
     result = "dieci";
   else if(dx == 1)
     result = "undici";
   else if(dx == 2)
     result = "dodici";
   else if(dx == 3)
     result = "tredici";
   else if(dx == 4)
     result = "quattordici";
   else if(dx == 5)
     result = "quindici";
   else if(dx == 6)
     result = "sedici";
   else if(dx == 7)
     result = "diciassette";
   else if(dx == 8)
     result = "diciotto";
   else if(dx == 9)
     result = "diciannove";
   }
  return result;
 }
/**Metodo che converte un numero compreso tra 0 e 9 dalla forma in cifre a
quella in caratteri.
@return Il numero convertito.*/
 public String unitàToString()
 {
 String result = "";
 if (number == 0)
   result = "";
 else if (number == 1)
   result = "uno";
 else if (number == 2)
   result = "due";
 else if (number == 3)
   result = "tre";
 else if (number == 4)
   result = "quattro";
 else if (number == 5)
   result = "cinque";
 else if (number == 6)
   result = "sei";
 else if (number == 7)
   result = "sette";
 else if (number == 8)
   result = "otto";
 else if (number == 9)
   result = "nove";
 return result;
 }
}
```

```
import java.util.Scanner;
public class NumberConverterTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.print("Inserire un numero compreso tra 0 e 999999 da convertire in
caratteri:");
  int num = in.nextInt();
  if (num < 0 || num > 999999)
    return;
  else
  {
   NumberConverter numero = new NumberConverter(num);
   System.out.println("Il numero è: " + numero.arabicToString());
  }
 }
}
```

Esercizio 5.6 – NumberConverterStatic

Testo:

Come nell'esercizio precedente si deve scrivere un programma che converta un numero nell'intervallo [1, 999.999] in una stringa con il nome del numero in italiano. In questo caso però si preferisce non creare l'oggetto della classe NumberConverter, ma tradurre il numero fornendolo come parametro esplicito.

Consigli:

La peculiarità di questo programma è che tutti i metodi sono static. Ricordare inoltre che quando vengono chiamati i metodi senza parametro implicito, Java considera automaticamente come se fosse inserito this.ilMioMetodo().

```
public class NumberConverterStatic
{
/**Metodo che converte il numero dividendo il problema in casi più
semplici invocando metodi ausiliari.
@param n Il numero tra 0 e 999.999 da convertire in lettere.
@return Il numero convertito in lettere.*/
 public static String arabicToString(int n)
 {
  String result = "";
  if (n == 0)
    result = "zero";
  else
  {
   int sin = n / 1000;
   int dex = n % 1000;
   if (sin == 0)
     result = centinaiaToString(dex);
   else if (sin == 1)
     result = "mille" + centinaiaToString(dex);
   else
     result = centinaiaToString(sin)+"mila"+centinaiaToString(dex);
  }
  return result;
 }
/**Si converte il numero dividendo il problema in casi più semplici ed
invocando metodi ausiliari.
@param n Il numero tra 1 e 999 da convertire in lettere.
@return Il numero convertito in lettere.*/
```

```
public static String centinaiaToString (int n)
 {
  String result = "";
  int sin = n / 100;
  int dex = n % 100;
  if (sin == 0)
    result = decineToString(dex);
  else if (sin == 1)
    result = "cento" + decineToString(dex);
  else
    result = unitàToString(sin) + "cento" + decineToString(dex);
  return result;
 }
/**Metodo che converte il numero dividendo il problema in casi più
semplici ed invocando metodi ausiliari.
@param n Il numero tra 1 e 99 da convertire in lettere.
@return Il numero convertito in lettere.*/
 public static String decineToString (int n)
 {
  String result = "";
  int sin = n / 10;
  int dex = n % 10;
  if (sin == 0)
  result = unitàToString(dex);
  else if (sin == 1)
  {
   if(dex == 0)
     result = "dieci";
   else if(dex == 1)
     result = "undici";
   else if(dex == 2)
     result = "dodici";
   else if(dex == 3)
     result = "tredici";
   else if(dex == 4)
     result = "quattordici";
   else if(dex == 5)
     result = "quindici";
   else if(dex == 6)
     result = "sedici";
   else if(dex == 7)
     result = "diciassette";
   else if(dex == 8)
     result = "diciotto";
   else if(dex == 9)
     result = "diciannove";
  }
  else if(sin == 2)
    result = "venti" + unitàToString(dex);
  else if(sin == 3)
    result = "trenta" + unitàToString(dex);
  else if(sin == 4)
    result = "quaranta" + unitàToString(dex);
  else if(sin == 5)
    result = "cinquanta" + unitàToString(dex);
  else if(sin == 6)
    result = "sessanta" + unitàToString(dex);
  else if(sin == 7)
    result = "settanta" + unitàToString(dex);
  else if(sin == 8)
    result = "ottanta" + unitàToString(dex);
  else if(sin == 9)
    result = "novanta" + unitàToString(dex);
```

```
  return result;
 }
/** Metodo che converte il numero dividendo il problema in casi più
semplici ed invocando metodi ausiliari.
@param n Il numero tra 1 e 9 da convertire in lettere.
@return Il numero convertito in lettere.*/
 public static String unitàToString(int n)
 {
  String result = "";
  if(n == 1) result = "uno";
  else if(n == 2)
    result = "due";
  else if(n == 3)
    result = "tre";
  else if(n == 4)
    result = "quattro";
  else if(n == 5)
    result = "cinque";
  else if(n == 6)
    result = "sei";
  else if(n == 7)
    result = "sette";
  else if(n == 8)
    result = "otto";
  else if(n == 9)
    result = "nove";
  return result;
 }
}
```

```
import java.util.Scanner;
public class NumberConverterStaticTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.print("Scrivi un numero intero tra 0 e 999.999 da scrivere in
lettere: ");
  int numero = in.nextInt();
  if (numero < 0 || numero > 999999)
    return;
  else
    System.out.println("Il numero è:
"+NumberConverterStatic.arabicToString(numero));
 }
}
```

Esercizio 5.7 – ArabicToRoman

Testo:

Realizzare una classe con un metodo che converta un numero intero compreso tra 1 e 100 in un numero romano (es: 57 diventa LVII).

Consigli:

L'esercizio non presenta particolari difficoltà. Si trattano separatamente le centinaia, le decine e le unità, e si associano al relativo simbolo romano.

```
public class ArabicToRoman
{
private int arabic;
/**Costruttore che inizializza il numero da convertire.
@param aNumber Il numero da convertire.*/
 public ArabicToRoman(int aNumber)
 {
  arabic = aNumber;
 }
/**Metodo che restituisce il numero tra 1 e 100 convertito in simboli romani.
@return Il numero convertito in simboli romani.*/
 public String converti()
 {
  String numeroRomano = "";
  int decine = arabic / 10;
  int unità = arabic % 10;
  if (decine == 10)
    return "C";
  if (unità == 1 )
    numeroRomano = "I";
  else if (unità == 2 )
    numeroRomano = "II";
  else if (unità == 3 )
    numeroRomano = "III";
  else if (unità == 4 )
    numeroRomano = "IV";
  else if (unità == 5 )
    numeroRomano = "V";
  else if (unità == 6 )
    numeroRomano = "VI";
  else if (unità == 7 )
    numeroRomano = "VII";
  else if (unità == 8 )
    numeroRomano = "VIII";
  else if (unità == 9 )
    numeroRomano = "IX";
  if (decine == 1 )
    numeroRomano = "X" + numeroRomano;
  else if (decine == 2 )
    numeroRomano = "XX" + numeroRomano;
  else if (decine == 3 )
    numeroRomano = "XXX" + numeroRomano;
  else if (decine == 4 )
    numeroRomano = "XL" + numeroRomano;
  else if (decine == 5 )
    numeroRomano = "L" + numeroRomano;
  else if (decine == 6 )
    numeroRomano = "LX" + numeroRomano;
  else if (decine == 7 )
    numeroRomano = "LXX" + numeroRomano;
  else if (decine == 8 )
    numeroRomano = "LXXX" + numeroRomano;

  else if (decine == 9 )
    numeroRomano = "XC" + numeroRomano;
  return numeroRomano;
 }
}
```

```
import java.util.Scanner;
public class ArabicToRomanTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.print("Inserire un numero tra 1 e 100 da convertire: ");
  int numero = in.nextInt();
  if (numero <= 0 || numero > 100)
    return;
  else
  {
   ArabicToRoman arabic = new ArabicToRoman(numero);
   System.out.println("Il numero richiesto è: "+arabic.converti());
  }
 }
}
```

Esercizio 5.8 – ArabicToRomanAvanzato

Testo:

Realizzare un convertitore arabo – romano che sfrutti tutti i simboli romani disponibili nel codice ASCII (I – V – X – L – C – D – M). Con questi simboli si possono scrivere tutti i numeri compresi tra 1 e 3999.

Consigli:

Il valore del numero è la somma dei valori dei caratteri. I è 1, II è 2, e III è 3. VI è 6 ("5 e 1"), VII è 7 e VIII è 8. I "caratteri di decina" (I, X, C, e M) possono esere ripetuti fino a tre volte. Alla quarta, si deve sottrarre uno dal più vicino "carattere di quintina" (V, L, D). Non si può rappresentare 4 come IIII, lo si deve rappresentare con IV ("1 in meno di 5"). 40 è scritto come XL, 41 come XLI, 42 come XLII, 43 come XLIII ed infine 44 come XLIV ("10 in meno di 50, più uno in meno di 5"). Similmente, arrivati al 9, si deve sottrarre dal "carattere di decina" immediatamente superiore. Ovvero una cifra inferiore scritta a sinistra di una cifra con il valore immediatamente maggiore si sottrae. 8 è VIII, ma 9 è IX ("uno in meno di dieci"), non VIIII (in quanto il carattere I non può essere ripetuto quattro volte). 90 è XC, 900 è CM. I "caratteri di quintina" non possono essere ripetuti. 10 è sempre rappresentato come X, mai come VV. 100 è sempre C, mai LL. Le cifre dei numeri romani sono sempre scritte dal più grande al più piccolo (ordine decrescente) e letti da sinistra a destra, per cui l'ordine dei caratteri è molto importante. DC è 600; CD è un numero completamente diverso (400, "100 meno di 500"). CI è 101; IC non è un numero romano valido (perché non si può sottrarre 1 direttamente da 100; 99 si deve scrivere XCIX, "10 in meno di 100 e poi 1 in meno di 10").

```
public class ArabicToRomanAvanzato
{
/**Metodo che converte il numero arabo in numero romano.
@param numeroArabo Il numero da convertire.
@return La conversione in numero romano.*/
 public static String converti(int numeroArabo)
 {
  String numeroRomano = "";
  int i = 1;
  int cinquina = 0;
  while (numeroArabo != 0)
  {    int resto = numeroArabo % 5;
   numeroArabo -= resto;
   if (numeroArabo % 10 == 5)
   {
    cinquina = 1;
    numeroArabo -= 5;
   }
```

```
   if (resto >= 0 && resto <= 3)
   {
    int n = resto;
    while (n > 0)
    {
     numeroRomano=getSimbolo(i)+numeroRomano;
     n -- ;
    }
   if (cinquina == 1)
     numeroRomano=getSimbolo(i+cinquina)+numeroRomano;
   }
   if (resto == 4)
   {
    if (cinquina == 1)
      numeroRomano=getSimbolo(i)+getSimbolo(i + 2)+numeroRomano;
    else
      numeroRomano=getSimbolo(i)+getSimbolo(i + 1)+numeroRomano;
   }
   cinquina = 0;
   i = i + 2;
   numeroArabo = Math. round(numeroArabo / 10);
  }
  return numeroRomano;
 }
/**Metodo ausiliario che restituisce i simboli romani.
@param posizione Il riferimento al simbolo.
@return Il simbolo da inserire.*/
 private static String getSimbolo(int posizione)
 {
  if (posizione == 1)
    return "I";
  else if (posizione == 2)
    return "V";
  else if (posizione == 3)
    return "X";
  else if (posizione == 4)
    return "L";
  else if (posizione == 5)
    return "C";
  else if (posizione == 6)
    return "D";
  else if (posizione == 7)
    return "M";
  return "errore in getSimbolo";
 }
}



import java.util.Scanner;
public class ArabicToRomanAvanzatoTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.print("Scrivi un numero tra 1 e 3999 da convertire: ");
  int numero = in.nextInt();
  if (numero < 1 || numero > 3999)
    return;
  else
    System.out.println("Il numero richiesto è: "
   +ArabicToRomanAvanzato.converti(numero));
 }}
```

Esercizio 5.9 - BankWithPassword

Testo:

Lo scopo del programma è riadattare il programma dell'esercizio 3.1, introducendo il metodo public boolean controlPassword(int pass), che restituisce true se la password inserita è corretta, false altrimenti. Questo metodo deve essere utilizzato all'interno di altri metodi per verificare che la password sia corretta e dare il consenso per procedere alle operazioni di deposit, whithdraw e addInterest (presenti nell'esercizio BanckAccount 3.1)

Consigli:

Per usare il metodo controlPassword all'interno di deposit, whitdraw, addInterest è sufficiente richiamare il metodo controlPassword(inserendogli come parametro un int che sia una password da controllare). Si tenga presente che controlPassword restituisce un boolean. Non è obbligatorio che il metodo controlPassword sia posizionato nella classe prima dei metodi deposit, whitdraw, addInterest.

```
public class BankWithPassword
{
private double balance;
private int password;
/**Costruttore che costruisce un conto con saldo uguale a zero e password di
accesso numerica.
@param pass Password numerica di accesso al conto.*/
public BankWithPassword(int pass)
{
balance = 0;
password = pass;
}
/**Costruttore che costruisce un conto con un saldo assegnato e password
di accesso numerica.
@param initialBalance Il saldo iniziale.
@param pass Password di accesso al conto.*/
public BankWithPassword(double initialBalance, int pass)
{
balance = initialBalance;
password = pass;
}
/**Metodo che verifica la validità della password immessa.
@param pass È la password da verificare.
@return true se corretta, false se errata.*/
public boolean controlPassword(int pass)
{
if (pass == password)
  return true;
return false;
}
/**Metodo che versa denaro nel conto bancario.
@param pass Password del conto.
@param amount L'importo da versare.
@return result true se l'operazione è andata a buon fine.*/

public boolean deposit(int pass, double amount)
{
if(controlPassword(pass)==true)
 {
  balance += amount;
  return true;
 }
return false;
}
```

```
/**Metodo che preleva denaro dal conto bancario se è possibile.
@param pass È la password che deve essere verificata.
@param amount L'importo da prelevare.
@return result true se l'operazione è andata a buon fine.*/
public boolean withdraw(int pass, double amount)
{
if(balance < amount)
return false;

 if(controlPassword(pass)==true)
 {
  balance -= amount;
  return true;
 }
return false;
}
/**Metodo che applica un interesse sul conto.
@param pass È la password che deve essere verificata.
@param rate Il tasso d'interesse.*/
public boolean addInterest(int pass, double rate)
{
 if(controlPassword(pass)==true)
 {
  balance += (balance * rate) / 100;
  return true;
 }
return false;
}
/**Metodo che restituisce il valore del saldo attuale del conto bancario.
@return Il saldo attuale.*/
public double getBalance()
{
return balance;
}
/**Metodo che restituisce la password del conto.
@return La password del conto.*/
public int getPassword()
{
return password;
}
}

import java.util.Scanner;
public class BankWithPasswordTester
{
 public static void main(String[] args)
 {
  Scanner in=new Scanner(System.in);
  System.out.println("Inserire un numero che sarà password del conto
unicredit");
  int pass = in.nextInt();
  BankWithPassword unicredit = new BankWithPassword(pass);
  System.out.println("Il saldo del conto 1 è: " +
  unicredit.getBalance()+ "\nla password del conto 1 è: " +
  unicredit.getPassword());
  System.out.println("Inserire un numero che sarà password del conto 2");
  int temp = in.nextInt();
  System.out.println("Inserire il saldo iniziale del conto 2");
  BankWithPassword sanPaolo = new BankWithPassword(in.nextDouble(),temp);
  System.out.println("Il saldo del conto 2 è: " +
  sanPaolo.getBalance()+ "\nla password del conto 2 è: " +
  sanPaolo.getPassword());
  System.out.println("Prelievo dal conto 2:\nInserire la password del conto 2");
```

```
  int password2 = in.nextInt();
  System.out.println("Inserire la somma da prelevare:");
  double amount2 = in.nextDouble();
  boolean result = sanPaolo.withdraw(password2, amount2);
  if (result)
    System.out.println("Operazione eseguita!");
  else
    System.out.println("Somma non disponibile nel conto o password errata");
  System.out.println("Versamento sul conto 1:\nInserire la password del conto  1
");
  int password1 = in.nextInt();
  System.out.println("Inserire la somma da versare:");
  double amount1 = in.nextDouble();
  boolean result2 = unicredit.deposit(password1, amount1);
  if(result)
    System.out.println("Operazione eseguita!");
  else
    System.out.println("Password errata. Impossibile effettuare il
versamento.");
  System.out.println("Nuovi saldi:\nconto 1: " +
  unicredit.getBalance() + "\nconto 2: " + sanPaolo.getBalance());
  unicredit.addInterest(password1, 3);
  sanPaolo.addInterest(password2, 3);
  System.out.println("Nuovi saldi con interesse applicato del 3%:");
  System.out.println("conto 1: " + unicredit.getBalance());
  System.out.println("\nconto 2: " + sanPaolo.getBalance());
 }
}
```

Esercizio 5.10 – CarFuel

Testo:

Modificare la classe Car2, assegnata nell'esercizio 3.4, in modo tale che la macchina riconosca il tipo di carburante che consuma(benzina o gasolio). Implementate inoltre i seguenti metodi: void aggiungiCarburante(double litri) che aggiunge carburante al serbatoio della macchina. void faiUnGiro(double kilometri) che simula una corsa di un'auto per un determinato numero di km. boolean usaBenzina() che restituisce se il carburante della macchina è benzina. boolean usaGasolio() che restituisce se il carburante della macchina è gasolio. double getSerbatoio()che restituisce il numero di litri di carburante rimanenti nel serbatoio. String getTipoDiCarburante() che restituisce il tipo di carburante della macchina.

Consigli:

L'esercizio non presenta particolari difficoltà.

```
public class CarFuel
{
private double serbatoio;
private boolean benzina;
private double resa;
/**Cosruttore che crea una macchina e gli assegna un serbatoio.
@param unCarburante tipo di carburante.
@param unaResa Rendimento dell'auto.
@param litri Litri presenti nel serbatoio dell'auto.*/
 public CarFuel(String unCarburante, double unaResa, double litri)
 {
  serbatoio = litri;
  resa = unaResa;
  if (unCarburante.equalsIgnoreCase("benzina")==true)
    benzina = true;
  else
    benzina = false;
  }
```

```
/**Metodo che aggiunge carburante al serbatoio della macchina.
@param litri Litri di carburante.*/
 public void aggiungiCarburante(double litri)
 {
  serbatoio += litri;
 }
/**Metodo che simula una corsa di un'auto per un determinato numero di km.
@param kilometri Kilimetri percorsi. */
 public void faiUnGiro(double kilometri)
 {
  double carburanteConsumato = kilometri/resa;
  if(carburanteConsumato <= serbatoio)
    serbatoio -= carburanteConsumato;
 }
/**Metodo che restituisce se il carburante della macchina è benzina.
@return Vero se l'auto è benzina.*/
 public boolean usaBenzina()
 {
  if (benzina==true)
    return true;
  else
    return false;
 }
/**Metodo che restituisce se il carburante della macchina è gasolio.
@return Tero se l'auto è gasolio.*/
 public boolean usaGasolio()
 {
  if (benzina==false)
    return true;
  else
    return false;
 }
/**Metodo che restituisce il numero di litri di carburante rimanenti nel
serbatoio.
@return serbatoio Litri rimanenti.*/
 public double getSerbatoio()
 {
  return serbatoio;
 }
/**Metodo che restituisce il tipo di carburante della macchina.
@return carburanteRisultante Tipo di carburante.*/
 public String getTipoDiCarburante()
 {
  String carburanteRisultante;
  if (benzina==true)
  {
   carburanteRisultante = "benzina";
  }
  else
  {
   carburanteRisultante = "gasolio";
  }
  return carburanteRisultante;
 }}
```

Esercizio 5.11 – DistributoreCarFuel

Testo:
Modificare la classe DistributoreBenzina, assegnata nell'esercizio 3.3, in modo che il distributore abbia due pompe, una di benzina ed una di gasolio. Ciascun tipo di carburante ha il suo costo per litro. Conseguentemente, dovrete raddoppiare il metodo rifornisci: dovrete avere un metodo per rifornire la

benzina ed un metodo per rifornire il gasolio. Il metodo vendi deve essere modificato nel seguente modo: public void vendi(double euro, CarFuel unaAutomobile). Tale metodo rifornisce l'automobile specificata come parametro esplicito nel modo appropriato (benzina oppure gasolio). Nel Tester create varie automobili e distributori. Fate compiere alcuni "viaggi" alle automobili, e rifornitele di benzina in modo appropriato. Inoltre, rifornite le pompe di benzina in caso di necessita'.

Consigli:
L'unico metodo difficoltoso è vendi, in quanto viene dato come parametro esplicito un oggetto della classe CarFuel (esercizio 5.10). Per sapere con quale combustibile funziona l'auto e per rifornirla si devono applicare i metodi della classe a cui appartiene l'oggetto. Quindi si useranno il metodo usaBenzina() e il metodo aggiungiCarburante(e come parametro un double che sia il numero di litri da aggiungere al serbatoio), della classe CarFuel.

```
public class DistributoreCarFuel
{
private double depositoBenzina;
private double depositoGasolio;
private double euroPerLitroDiBenzina;
private double euroPerLitroDiGasolio;
/**Costruttore che crea un distributore.
@param tariffaBenzina Prezzo al litro di benzina.
@param tariffaGasolio Prezzo al litro di gasolio.*/
 public DistributoreCarFuel(double tariffaBenzina, double tariffaGasolio)
 {
  euroPerLitroDiGasolio = tariffaGasolio;
  euroPerLitroDiBenzina = tariffaBenzina;
  depositoBenzina=0;
  depositoGasolio=0;
 }
/**Metodo che aggiunge litri di benzina.
@param litri Litri di benzina aggiunti.*/
 public void aggiungiBenzina(double litri)
 {
  depositoBenzina += litri;
 }
/**Metodo che aggiunge litri di gasolio.
@param litri Litri di gasolio aggiunti.*/
 public void aggiungiGasolio(double litri)
 {
  depositoGasolio += litri;
 }
/**Metodo che simula la vendita di carburante.
@param euro Euro di benzina che vengono venduti.
@param miaAuto La macchina a cui vengono venduti.*/
public void vendi(double euro, CarFuel miaAuto)
 {
  double litriDaTogliereBenzina = 0;
  double litriDaTogliereGasolio = 0;
  if (miaAuto.usaBenzina()==true)
  {
   litriDaTogliereBenzina = euro / euroPerLitroDiBenzina;
   if (litriDaTogliereBenzina <= depositoBenzina)
   {
    depositoBenzina -= litriDaTogliereBenzina;
    miaAuto.aggiungiCarburante(litriDaTogliereBenzina);
   }
  }
```

```
else
  {
   litriDaTogliereGasolio = euro / euroPerLitroDiGasolio;
   if (litriDaTogliereGasolio <= depositoGasolio)
   {
    depositoGasolio -= litriDaTogliereGasolio;
    miaAuto.aggiungiCarburante(litriDaTogliereGasolio);
   }
  }
 }

/**Metodo che reimposta i prezzi dei carburanti.
@param unaTariffaBenzina Prezzo al litro di benzina.
@param unaTariffaGasolio Prezzo al litro di gasolio.*/
 public   void   reimpostaPrezziCarburanti(double   unaTariffaBenzina,   double
unaTariffaGasolio)
 {
  euroPerLitroDiGasolio = unaTariffaGasolio;
  euroPerLitroDiBenzina = unaTariffaBenzina;
 }
/**Metodo che restituisce quanta benzina rimane.
@return depositoBenzina Quantità di benzina nel deposito.*/
 public double getLitriBenzina()
 {
  return depositoBenzina;
 }
/**Metodo che restituisce quanto gasolio rimane.
@return depositoGasolio Quantità di gasolio nel deposito.*/
 public double getLitriGasolio()
 {
  return depositoGasolio;
 }
/**Metodo che restituisce il prezzo attuale della benzina.
@return euroPerLitroDiBenzina Prezzo della benzina.*/
 public double getPrezzoBenzina()
 {
  return euroPerLitroDiBenzina;
 }
/** Metodo che restituisce il prezzo attuale del gasolio.
@return euroPerLitroDiGasolio Prezzo del gasolio.*/
 public double getPrezzoGasolio()
 {
  return euroPerLitroDiGasolio;
 }
}
```

```
import java.util.Scanner;
public class DistributoreCarFuelTester
{
 public static void main (String [] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Programma che simula 2 distributori e delle auto che si
rifornisconо");
  System.out.println("Inserire i prezzi dei carburanti Esso e Agip:");
  System.out.print("Euro al litro di Benzina Esso: ");
  double prezzoBenzinaEsso = in.nextDouble();
  System.out.print("Euro al litro di Gasolio Esso: ");
  double prezzoGasolioEsso = in.nextDouble();
```

```
  System.out.print("Euro al litro di Benzina Agip: ");
  double prezzoBenzinaAgip = in.nextDouble();
  System.out.print("Euro al litro di Gasolio Agip: ");
  double prezzoGasolioAgip = in.nextDouble();
/**Creo i due distributori.*/
  DistributoreCarFuel esso = new DistributoreCarFuel(prezzoBenzinaEsso,
prezzoGasolioEsso);
  DistributoreCarFuel agip = new DistributoreCarFuel(prezzoBenzinaAgip,
prezzoGasolioAgip);
/**Rifornisco i depositi dei distributori.*/
  System.out.println("Numero di litri di rifornimento al deposito di Benzina
Esso:");
  double litriBenzinaEsso = in.nextDouble();
  esso.aggiungiBenzina(litriBenzinaEsso);
  System.out.println("Numero di litri di rifornimento al deposito di Gasolio
Esso:");
  double litriGasolioEsso = in.nextDouble();
  esso.aggiungiGasolio(litriGasolioEsso);
  System.out.println("Numero di litri di rifornimento al deposito di Benzina
Agip:");
  double litriBenzinaAgip = in.nextDouble();
  agip.aggiungiBenzina(litriBenzinaAgip);
  System.out.println("Numero di litri di rifornimento al deposito di Gasolio
Agip:");
  double litriGasolioAgip = in.nextDouble();
  agip.aggiungiGasolio(litriGasolioAgip);
/**Stampo la situazione dei depositi.*/
  System.out.println("Nel distributore Esso ci sono "+esso.getLitriBenzina()+"
litri di benzina");
  System.out.println("Nel distributore Esso ci sono "+esso.getLitriGasolio()+"
litri di gasolio");
  System.out.println("Nel distributore Agip ci sono "+agip.getLitriBenzina()+"
litri di benzina");
  System.out.println("Nel distributore Agip ci sono "+agip.getLitriGasolio()+"
litri di gasolio");
/**Creo un'auto, la chiamo Ferrari.*/
  System.out.print("Inserire dati dell'auto(Ferrari): ");
  System.out.println("Inserire il carburante di alimento dell'auto(benzina o
gasolio) ");
  String sceltaFerrari = in.next();
  System.out.println("Inserire il rendimento dell'auto(qunati Km fa con 1
litro)");
  double resaFerrari = in.nextDouble();
  CarFuel ferrari = new CarFuel(sceltaFerrari, resaFerrari, 0);
  System.out.print("Inserire numero euro di carburante che si desidera
acquistare dal distributore Esso: ");
  double euroCarburante = in.nextDouble();
  esso.vendi(euroCarburante, ferrari);
  System.out.println("Nel distributore Esso ora ci sono
"+esso.getLitriGasolio()+" litri di gasolio");
  System.out.println("Nel distributore Esso ora ci sono
"+esso.getLitriBenzina()+" litri di benzina");
  System.out.println("Nel serbatoio della Ferrari ci sono
"+ferrari.getSerbatoio()+" litri di carburante");
/**Creo un'auto, la chiamo Maserati.*/
  System.out.print("Inserire dati dell'auto(Maserati): ");
  System.out.print("Inserire il carburante di alimento dell'auto(benzina o
gasolio) ");
  String sceltaMaserati = in.next();
  System.out.println("Inserire il rendimento dell'auto(qunati Km fa con 1
litro)");
  double resaMaserati = in.nextDouble();
  CarFuel maserati = new CarFuel(sceltaMaserati, resaMaserati, 0);
```

```
  System.out.print("Inserire numero euro di carburante che si desidera
acquistare  dal distributore Agip: ");
  euroCarburante = in.nextDouble();
  agip.vendi(euroCarburante, maserati);
  System.out.println("Nel      distributore      Agip      ora      ci      sono
"+agip.getLitriGasolio()+" litri di gasolio");
  System.out.println("Nel      distributore      Agip      ora      ci      sono
"+agip.getLitriBenzina()+" litri di benzina");
  System.out.println("Nel    serbatoio    della    Maserati    ci    sono    "
+maserati.getSerbatoio()+" litri di carburante");
/**Rifornisco i depositi dei distributori.*/
  System.out.println("Ora aggiungiamo litri ai depositi dei distributori Esso e
Agip:");
  System.out.println("Numero di litri di rifornimento al deposito di Benzina
Esso:");
  litriBenzinaEsso = in.nextDouble();
  esso.aggiungiBenzina(litriBenzinaEsso);
  System.out.println("Numero di litri di rifornimento al deposito di Gasolio
Esso:");
  litriGasolioEsso = in.nextDouble();
  esso.aggiungiGasolio(litriGasolioEsso);
  System.out.println("Numero di litri di rifornimento al deposito di Benzina
Agip:");
  litriBenzinaAgip = in.nextDouble();
  agip.aggiungiBenzina(litriBenzinaAgip);
  System.out.println("Numero di litri di rifornimento al deposito di Gasolio
Agip:");
  litriGasolioAgip = in.nextDouble();
  agip.aggiungiGasolio(litriGasolioAgip);
/**Stampo la situazione dei depositi.*/
  System.out.println("Nel distributore Esso ci sono "+esso.getLitriBenzina()+"
litri di benzina");
  System.out.println("Nel distributore Esso ci sono "+esso.getLitriGasolio()+"
litri di gasolio");
  System.out.println("Nel distributore Agip ci sono "+agip.getLitriBenzina()+" l
litri di benzina");
  System.out.println("Nel distributore Agip ci sono "+agip.getLitriGasolio()+"
litri di gasolio");
/**Faccio compiere dei viaggi alle auto.*/
  System.out.print("Inserire il numero di Km da percorrere in Ferrari: ");
  double numeroKm = in.nextDouble();
  ferrari.faiUnGiro(numeroKm);
  System.out.println("Nel    serbatoio    della    Ferrari    ora    ci    sono
"+ferrari.getSerbatoio()+" litri di carburante");
  System.out.print("Inserire il numero di Km da percorrere in Maserati: ");
  numeroKm = in.nextDouble();
  maserati.faiUnGiro(numeroKm);
  System.out.println("Nel    serbatoio    della    Maserati    ora    ci    sono
"+maserati.getSerbatoio()+" litri di carburante");
 }}
```

CAPITOLO 6 Iterazioni

REMIND:

(Paragrafo 6.1) WHILE

- while → esegue l'enunciato finché la condizione è vera.
- do…while → esegue l'enunciato fintanto che non si verifica la condizione.
- boolean done = false; while(!done) {} → done è una 'flag' (bandierina).

(Paragrafo 6.2) for

- for(inizializzazione; condizione; aggiornamento) → ripete l' esecuzione di un enunciato e aggiorna un espressione finchè una condizione è vera.
- for innestati → uno dentro l'altro, solitamente si usano per elaborare strutture bidimensionali.

Esercizio 6.1 – Fact

Testo:

Scrivere un programma che calcoli il fattoriale di un numero assegnato n.

Consigli:

La difficoltà maggiore riguarda l'implementazione in codice del fattoriale, ma questa difficoltà può esser risolta con il seguente ragionamento. Si crea un metodo, conv(), nel quale si creano due variabili locali: i (valore temporaneo) e result (il risultato finale dell'operazione). Fintanto che i è minore o uguale al valore assegnato dall'utente, si moltiplica i per il risultato, e la variabile i viene incrementata. L'operazione si concluderà solo quando i sarà pari a n, ed il metodo stamperà il risultato finale.

```
public class Fact
{
private int n;
/**Si costruisce un calcolatore per il fattoriale.
@param k Il numero di partenza.*/
 public Fact(int k)
 {
  n = k;
 }
/**Si calcola il fattoriale.
@return result Il risultato.*/
 public long conv()
 {
  int i = 1;
  long result = 1;
  while (i <= n)
  {
   result = result * i;
   i++;
  }
  return result;
  }
 }
```

```
import java.util.Scanner;
public class FactTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Inserisci il numero del quale vuoi calcolare il
fattoriale:");
  int a = in.nextInt();
  Fact n = new Fact(a);
  System.out.println(n.conv());
 }
}
```

Esercizio 6.2 – PrimeGenerator

Testo:

Scrivere un programma che chieda all'utente un numero intero e che poi stampi tutti i numeri primi fino al numero stesso. Per esempio, se l'utente immette 20, il programma stamperà:

2
3
5
7
11
13
17
19

Ricordarsi che un numero è primo se non è divisibile per nessun altro numero, salvo 1 e se stesso. Usate una classe PrimeGenerator con un metodo nextPrime.

Consigli:

Le maggiori difficoltà dell'esercizio possono essere riscontrate nel metodo boolean isPrime(int a), il quale deve definire se un dato numero in ingresso è primo. Si creino due variabili locali: una boolean, definita primo, che restituisce true se il numero è primo (funziona da sentinella), l'altra int, indicata con i, che serve da denominatore per verificare se la divisione tra il numero dato in ingresso ed i risulta zero. Fintanto che i è minore o pari al dato in ingresso e la divisione tra il numero stesso e i dà resto, allora s'incrementa la variabile i. Se risulta che i è pari al numero dato, ciò significa che è primo (un numero si definisce primo se è divisibile solo per se stesso e per 1), e quindi il programma restituisce true, altrimenti false.
Il metodo String getFirstPrimes(int input), invece, stampa tutti i numeri primi compresi tra 1 e il dato iniziale: ciò viene eseguito con un ciclo for, che restituisce i valori sottoforma di stringhe.

```
public class PrimeGenerator
{
private int n;
/**Si costruisce un generatore di numeri primi.
@param n Il numero fino a cui devo trovare i primi.*/
 public PrimeGenerator(int numero)
 {
  n = numero;
 }
/**Si verifica se un numero è primo.
@param a Il numero su cui eseguo il test.
@return Il risultato del test.*/
 public boolean isPrime(int a)
 {
  if (a <= 1)
    return false;
  int i = 2;
  while (i <= a / 2)
  {
   if (a % i == 0)
     return false;
   i++;
  }
  return true;
 }

/**Stampa i numeri primi.
 @return primi I numeri primi.*/
```

```
public String getFirstPrimes()
 {
  String primi = "";
  for (int i=2; i <= n; i++)
  {
   if (isPrime(i))
     primi = primi + i + "\n";
  }
  return primi;
 }
}
```

```
import java.util.Scanner;
public class PrimeGeneratorTester
{
 public static void main(String[] args)
 {
  Scanner console = new Scanner(System.in);
  System.out.println("Inserisci il numero fino a cui devo trovare i primi");
  int n = console.nextInt();
  PrimeGenerator max = new PrimeGenerator(n);
  System.out.println(max.getFirstPrimes());
 }
}
```

Esercizio 6.3 – TextStatistic

Testo:
Scrivere un programma che sia in grado di verificare la presenza di una parola all'interno di un testo.

Consigli:
Il metodo stat(String word) funziona da "motore di ricerca": se nel testo compare la parola ricercata, s'incrementa la variabile locale result, che indica il numero di ripetizioni del termine dato in ingresso. Alla fine, esso restituirà il numero totale di volte con cui la parola cercata è presente nel testo.

```
public class TextStatistic
{
private String text;
/**Si costruisce un oggetto che cerca in una stringa.
@param unTesto La stringa su cui cercare.*/
 public TextStatistic(String unTesto)
 {
  text = unTesto;
 }
/**Si conta quante volte una parola compare in un testo.
@param word La parola da cercare.
@return Quante volte la parola cercata compare.*/
 public int stat(String word)
 {
  int result = 0;
  for (int i = 0; i <= (text.length() - word.length()); i++)
  {
  if(text.substring(i,i+word.length()).toUpperCase().equals(word.toUpperCase()))
    result++;
  }
  return result;
 }
```

```
import java.util.Scanner;
public class TextStatisticTester
{
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);
  System.out.println("Immettere il testo sul quale si desidera effettuare la
ricerca");
  String text = input.nextLine();
  TextStatistic times = new TextStatistic(text);
  System.out.println("Immettere la parola che si desidera cercare");
  String word = input.nextLine();
  System.out.println(times.stat(word));
 }
}
```

Esercizio 6.4 - FibonacciGenerator

Testo:

Scrivere un programma che calcoli l'n-esimo numero della successione di Fibonacci.

Consigli:

L'esercizio non presenta particolari difficoltà. Si realizza il programma creando un convertitore non universale, in cui il numero n specificato in ingresso dal tester viene incorporato nella classe sotto forma di variabile d'istanza non successivamente modificabile. Si ricorda che la successione di Fibonacci è una sequenza di numeri interi naturali ciascuno dei quali è il risultato della somma dei due precedenti (0,1,1,2,3,5,8…).

```
public class FibonacciGenerator
{
private int indice;
/**Creo un oggetto della classe FibonacciGenerator.
 @param unNumero N-esimo numero della serie di Fibonacci.*/
 public FibonacciGenerator(int unNumero)
 {
  indice = unNumero;
 }
/**Caclola l'n-esimo numero della serie di Fibonacci.*/
 public long getFibonacci()
 {
  long result = 0;
  long temp1 = 0;
  long temp2 = 1;
  for (int i = 0; i < indice; i++)
  {
   result = temp1 + temp2;
   temp1 = temp2;
   temp2 = result;
  }
  return result;
 }
}
```

```
import java.util.Scanner;
public class FibonacciGeneratorTester
{
 public static void main (String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Inserire un numero intero positivo: ");
```

```
  if (in.hasNextInt())
  {
   int indice = in.nextInt();
  if (indice < 0)
    return;
  else
  {
   FibonacciGenerator fib = new FibonacciGenerator(indice);
   System.out.println("Il numero di Fibonacci cercato è " + fib.getFibonacci());
  }
  }
 }
}
```

Esercizio 6.5 - FibonacciGeneratorStatic

Testo:

Scrivere un programma che calcoli l'n-esimo numero della serie di Fibonacci.

Consigli:

In questa seconda versione si realizza il programma creando un metodo statico. In questo modo, il programma puo' essere utilizzato per diverse conversioni.

```
/*Creo una classe con un metodo statico che prende un numero intero n
 * e restituisce l'n-esimo numero della serie di Fibonacci*/
public class FibonacciGeneratorStatic
{
/**Creo un metodo statico che prende un numero intero n
 e restituisce l'n-esimo numero della serie di Fibonacci.
 @param aNamber Numero intero n.*/
 public static long getFibonacciAt(int aNumber)
 {
  long result = 0;
  long temp1 = 0;
  long temp2 = 1;
  for (int i = 0; i < aNumber; i++)
  {
   result = temp1 + temp2;
   temp1 = temp2;
   temp2 = result;
  }
  return result;
  }
 }
```

```
import java.util.Scanner;
public class FibonacciGeneratorStaticTester
{
 public static void main (String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Inserire un numero intero positivo: ");
  int n = in.nextInt();
  System.out.println("Il numero di Fibonacci cercato è "
  + FibonacciGeneratorStatic.getFibonacciAt(n));
 }
}
```

CAPITOLO 7 Vettori e Array

REMIND:

(Paragrafo 7.1) ARRAY

- array → sequenza di valori del medesimo tipo.
- i valori degli indici iniziano da 0 (zero)
- data.length → lunghezza dell'array data (notare: length non è seguito da parentesi).
- la loro lunghezza è fissa.

(paragrafo 7.2) VETTORI

- per utilizzarli è necessario importare la classe ArrayList.
- ArrayList, *vantaggi*:
  - un ArrayList può crescere e calare di dimensione.
  - ci sono metodi per svolgere le operazioni più comuni (inserimento, rimozione…).
- *Metodi*:
  - .add(a) → aggiunge un oggetto a alla fine della sequenza.
  - .add(i, c) → aggiunge nella posizione i l'oggetto c e sposta i successivi in avanti di una posizione.
  - .remove(i) → elimina l'elemento nella posizione i.
  - .size() → restituisce la dimensione dell'array.
  - .get(i) → restituisce l'oggetto nella posizione i.
  - .set(i, a) → assegna il valore a alla posizione i.
  - .isEmpty()→ controlla se l'Arraylist è vuoto(boolean).

(paragrafo 7.3) INVOLUCRI E AUTOIMPACCHETTAMENTO (wrapping)

- autoimpacchettamento → conversione tra tipi primitivi e le corrispondenti classi involucro: avviene automaticamente.
- auto-unboxing → gli oggetti involucro vengono tolti dalla scatola per generare valori di tipo primitivo.

(paragrafo 7.3) IL CICLO "FOR" GENERALIZZATO

- visita tutti gli oggetti di una sequenza in modo non necessariamente ordinato(dell'inizio alla fine o da sinistra a destra), questo dipende dall'elaboratore a cui è data da esegure l'istruzione.

```
es: double[] data = ...;
 double sum = 0;
 for (double e: data)
 { sum = sum + e }
```

(paragrafo 7.7) COPIARE ARRAY

- clone → vera copia di un array
  `double[] prices = (double[]) data.clone();`
- copiare elementi da un array a un altro
  → `System.arraycopy(from, fromStart, to , toStart, count);`
- aggiungere un nuovo elemento nella posizione i nell'array data
  → `System.arraycopy(data, i, data, i+1, data.length-i-1); data[i] = x;`
- rimuovere un elemento nella posizione i nell'array data
  → `System.arraycopy(data, i+1, data, i, data.length-i-1);`

Esercizio 7.1 – cancArrayList

Testo:

Realizzare una classe cancArrayList che cancelli i numeri pari presenti in un ArrayList di interi. Tale classe presenta una sola variabile di istanza private ArrayList<Integer> lista, creare quindi un metodo void inserisci(int n) che inserisca nell'ArrayList lista il numero intero passato come parametro esplicito, creare un metodo void cancLista() che cancelli dall'ArrayList lista tutti gli interi pari, implementare infine un metodo String stampa che restituisce sottoforma di stringa il contenuto dell' ArrayList lista.

Consigli:

Si raccomanda di prestare molta attenzione al metodo cancLista(), poichè essendo lista un ArrayList, ogni volta che un oggetto viene eliminato, tutti gli oggetti successivi vengono spostati automaticamente (il loro indice viene decrementato di una unità). Nel nostro caso se ci troviamo ad avere più interi pari consecutivi nall'ArrayList lista non tutti verranno cancellati(se usiamo un for che visita l'ArrayList dall'inizio alla fine). Ci sono solo due modi per cancellare con successo più elementi contigui da un ArrayList: (primo)fare in modo che il for visiti gli elementi dall'ultimo al primo; (secondo) ogni volta che un elemento viene cancellato decrementare l'indice di una unità. Si prega di implementare, per esercizio, due diversi metodi cancListaPrimo() e cancListaSecondo().

```
import java.util.ArrayList;
public class CancArrayList
{
private ArrayList<Integer> lista;
/**Creo un oggetto della classe cancArrayList.*/
 public CancArrayList()
 {
  lista=new ArrayList<Integer>();
 }
/**Aggiunge all'ArrayList gli interi.
 @param n Interi.*/
 public void inserisci(int n)
 {
  lista.add(n);
 }
/**Cancella dall'ArrayList gli interi pari(primo metodo).*/
 public void cancListaPrimo()
 {
  for(int i=lista.size()-1;i>-1;i--)
  {
   if(lista.get(i)%2==0)
     lista.remove(i);
  }
 }
/**Cancello dall'ArrayList gli interi pari(secondo metodo).*/
 public void cancListaSecondo()
 {
  for(int i=0;i<lista.size();i++)
  {
   if(lista.get(i)%2==0)
   {
    lista.remove(i);
    i--;
   }
  }
}
```

```
/**Restituisce il contenuto dell'ArrayList.
 @return Situazione ArrayList.*/
 public String stampa()
 {
  String r = "";
  r = "Interi presenti nell'Arraylist: "+lista;
  return r;
 }
}


import java.util.Scanner;
public class CancArrayListTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  CancArrayList r=new CancArrayList();
  String scelta = "y";
  while(scelta.equals("y"))
  {
   System.out.print("Inserire un numero intero: ");
   int i = in.nextInt();
   r.inserisci(i);
   System.out.print("Desideri inserire un altro numero?(y/n): ");
   scelta = in.next();
  }
  System.out.print("Desideri  cancellare  l'ArrayList  con  il  primo  o  secondo
metodo?(1/2): ");
  int fine = in.nextInt();
  if(fine==1)
  {
   r.cancListaPrimo();
   System.out.print(r.stampa());
  }
  if(fine==2)
  {
   r.cancListaSecondo();
   System.out.print(r.stampa());
  }
 }
}
```

Esercizio 7.2 – Purse

Testo:

Realizzare una classe Purse ("borsellino"), che simula una raccolta di monete. Per semplicità, si memorizzeranno solo i nomi delle monete in un ArrayList<String>.
Fornire i seguenti metodi: un construttore che che crea un oggetto Purse vuoto, un metodo public void addCoin(String coinName) che inserisce una moneta al purse e un metodo toString che stampa le monete presenti nel borsellino, nel formato seguente: Purse[Quarter, Dime, Nickel, Dime].

Consigli:

Si raccomanda d'importare la classe ArrayList, altrimenti Java non riesce ad individuare il comando "add" del metodo void addCoin(String coinName). La maggiore difficoltà si incontra nel metodo toString, dove si usa un ciclo for per visitare l'ArrayList coins e stamparne il contenuto.

```
import java.util.ArrayList;
public class Purse
{
private ArrayList<String> coins;

/**Crea un oggetto della classe Purse.*/
 public Purse()
 {
  coins = new ArrayList<String>();
 }
/**Aggiunge una moneta al purse.
@param coinName Il nome della moneta da aggiungere.*/
 public void addCoin(String coinName)
 {
  coins.add(coinName);
 }
/**Traduce in stringa il contenuto del purse.
@return result Il contenuto nel formato richiesto.*/
 public String toString()
 {
  if (coins.size() == 0)
    return "Empty purse!";
  else
    {
     String result = "Purse[";
     for (int i = 0; i < coins.size(); i++)
     {
      result = result + coins.get(i);
       if (i < coins.size() -1) result = result + ", ";
     }
   return result + "]";
 }
}
```

```
import java.util.Scanner;
public class PurseTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  Purse p = new Purse();
  boolean add = true;
  System.out.println("Inserire una moneta nel borsellino, oppure 0(zero) per
terminare.");
  while (add)
  {
   String s = in.next();
   if(s.equals("0"))
     add = false;
   else
     p.addCoin(s);
  }
  System.out.println(p.toString());
 }
}
```

Esercizio 7.3 – MinMaxArray

Testo:

Scrivere un programma che individui il valore massimo e minimo all'interno di un' array.

Consigli:

I metodi getMin() e getMax(), dal punto di vista dell'implementazione, sono praticamente identici: entrambi definiscono una variabile locale intera, a cui viene assegnata la prima posizione dell'array temporanea. Con un ciclo for, si visitano tutti i valori dell'array: se il valore della posizione successiva è maggiore o minore (a seconda del caso) di quello contenuto nella posizione 0, allora lo si sostituisce. Alla fine, il metodo restituisce il valore massimo o minimo.

```
public class MinMaxArray
{
private int[] pippo;
/**Costruisce un oggetto della classe MinMaxArray.
@param unVettore Il vettore che l'oggetto deve contenere.*/
 public MinMaxArray(int[] unVettore)
 {
  pippo = new int[unVettore.length];
  System.arraycopy(unVettore, 0, pippo, 0, unVettore.length);
 }
/**Restituisce il valore massimo contenuto nell'array.
@return Il valore massimo contenuto nell'array.*/
 public int getMax()
 {
  int massimo = pippo[0];
  for(int i= 0; i < pippo.length; i++)
  {
   if(massimo < pippo[i])
     massimo = pippo[i];
  }
 return massimo;
 }

/**Restituisce il valore minimo contenuto nell'array.
@return Il valore minimo contenuto nell'array.*/
 public int getMin()
 {
  int minimo = pippo[0];
  for(int i= 0; i < pippo.length; i++)
  {
   if(minimo > pippo[i])
   minimo = pippo[i];
  }
  return minimo;
 }
}
```

```
import java.util.Scanner;
public class MinMaxArrayTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Inserire la lunghezza del vettore da costruire:");
  int lunghezza = in.nextInt();
  if (lunghezza < 1)
    return;
  else
  {
   int[] myArray = new int[lunghezza];
   for(int i = 0; i < myArray.length; i++)
   {
    System.out.println("Inserire un intero da aggiungere nella posizione " + i);
    myArray[i] = in.nextInt();
   }
   MinMaxArray pluto = new MinMaxArray(myArray);
   System.out.println("numero minimo: ");
   System.out.println(pluto.getMin());
   System.out.println("numero massimo: ");
   System.out.println(pluto.getMax());
  }
 }
}
```

Esercizio 7.4 – Arcobaleno

Testo:

Progettare una classe di nome Arcobaleno, che abbia come variabili di istanza private String[ ] colori e private int numeroColori. I metodi richiesti sono: public Arcobaleno(int Max), Il costruttore inizializza le variabili di istanza, la lunghezza dell'array colori è passata come parametro esplicito max. public void aggiungiColore(String unColore), aggiunge un colore all'array colori. public void rimuoviColore(StringUnColore), elimina il colore dall'array colori. public int coloriVisibili() restituisce il numero di colori presenti nell'array colori.
L'array colori deve essere gestito come un array parzialmente riempito e non può contenere più colori uguali.

Consigli:

L'array gestito come parzialmente riempito significa che va riempito dell'indice zero in avanti in modo ordinato, non casuale, così da sapere dove sono gli i "cassetti" pieni e quelli vuoti. La variabile numeroColori segna il limite tra i "cassetti" pieni e quelli vuoti, viene chiamata anche sentinella, e precisamente indica il primo "cassetto" vuoto.
Visto che non ci possono essere più colori uguali nell'array è necessario fare un controllo ogni volta che un colore viene inserito: si verifica che non sia già presente nell'array.
Si consiglia implementare un metodo ausiliario private int cercaColore(String unColore) che cerca un colore nell'array colori, restituisce -1 se il colore non è presente, restituisce un numero apantenente all'intervallo [0, max] se il colore viene trovato, il numero ne indica anche la posizione nell'array colori.

```
public class Arcobaleno
{
 private String[] colori;
 private int numeroColori;//sentinella
 /**Crea un oggetto della classe Arcobaleno.
  @param max dimensione array colori.*/
```

```
    public Arcobaleno(int max)
    {
     colori= new String[max];
     numeroColori=0;
    }
    /**Metodo ausiliario che cerca un colore
     @param unColore colore da cercare.
     @return indice al quale si trova il colore cercato.*/
    private int cercaColore(String unColore)
    {
     int result=-1;
     for(int i=0;i<numeroColori;i++)
     {
      if(colori[i].equals(unColore))
            result=i;
     }
     return result;}
    /**Aggiunge un colore all'array colori.
     @param unColore colore da inserire.*/
    public void aggiungiColore(String unColore)
    {
     if(cercaColore(unColore)!=-1)
      return;
     if(numeroColori>=colori.length)
      return;
     colori[numeroColori]=unColore;
     numeroColori++;
    }
    /**Elimina un colore dall'array colori.
     @param unColore colore da eliminare.*/
    public void rimuoviColore(String unColore)
    {
     if(cercaColore(unColore)==-1)
      return;
     colori[cercaColore(unColore)]=colori[numeroColori-1];
     numeroColori--;
    }
    /**Restituisce il numero di colori presenti nell'array colori.
     @return numeroColori numero di colori presenti nell'array colori.*/
    public int coloriVisibili()
    {
     return numeroColori;
    }
}


public class ArcobalenoTester
{
  public static void main(String[] args)
  {
   Arcobaleno test=new Arcobaleno(5);
   test.aggiungiColore("rosso");
   test.aggiungiColore("giallo");
   test.aggiungiColore("blu");
   test.aggiungiColore("verde");
   test.aggiungiColore("viola");
   test.aggiungiColore("ciano");
   System.out.println(test.coloriVisibili());
   test.rimuoviColore("giallo");
   System.out.println(test.coloriVisibili());
  }
}
```

Esercizio 7.5 - Permutazioni

Testo:

Scrivete un programma che produca 10 permutazioni casuali dei numeri da 1 a 10. Per generare una permutazione casuale dovete riempire un array con i numeri da 1 a 10 in modo che siano tutti diversi. Potreste farlo in modo brutale, chiamando Random.nextInt finché produce un valore non ancora presente nell'array, ma dovreste, invece, usare un metodo più intelligente: create un secondo array e inseritevi i numeri da 1 a 10; poi, prendetene uno a caso, rimovetelo e accordatelo all'array che contiene la permutazione. Ripetete 10 volte. Realizzate una classe PermutationGenerator che abbia il metodo int[] nextPermutation.

Consigli:

Suggerimenti sullo sviluppo dell'esercizio sono già contenuti nel testo del programma.

```
import java.util.Random;
public class Permutazioni
{
private int[] permutazione;
private int[] numeri;
/**Crea un oggetto della classe Permutazioni.*/
 public Permutazioni()
 {
  permutazione = new int[10];
  numeri = new int[10];
  for (int i=0; i<numeri.length; i++)
  {
   numeri[i]=i + 1;
  }
 }
/**Inserisce nell'array permutazioni numeri casuali.*/
 public int[] nextPermutation()
 {
  Random genera = new Random();
  int cont = 0;
  while (cont < 10)
  {
   int temp1 = genera.nextInt(10);
   if((numeri[temp1] != 0))
   {
    permutazione[cont] = numeri[temp1];
    numeri[temp1] = 0;
    cont++;
   }
  }
  return permutazione;
 }
/**Trasforma l'array in una stringa.*/
 public String toString()
 {
  String s = "";
  for (int i=0; i<10; i++)
  {
   s = s + permutazione[i] + " ";
  }
  return s;
 }
}
```

```
public class PermutazioniTester
{
 public static void main(String[] args)
 {
  final int NUM_PERM = 10;
  for (int i = 0; i < NUM_PERM; i++)
  {
   Permutazioni p = new Permutazioni();
   int[] aPerm = p.nextPermutation();
   for (int j = 0; j < 10; j++)
    System.out.print(aPerm[j] + " ");
   System.out.println("\n");
  }
 }
 }
```

Esercizio 7.6 - Autorimessa

Testo:

Progettare una classe di nome Autorimessa, per rappresentare un parcheggio per automobili.
Tale classe prevede una variabile di istanza. La variabile e' definita come Arraylist<String> box, che rappresenta una lista con le targhe delle auto parcheggiate nella autorimessa.
La classe deve avere un costruttore public Autorimessa(), che inizializza l'ArrayList box. Un metodo definito come public void parcheggia(String unaTarga), che parcheggia un'auto. Un metodo definito come public void rimuovi(String unaTarga), che elimina l'auto con targa specificata dal parametro esplicito dal parcheggio. Un metodo public int numeroAuto(), che restituisce il numero di auto presenti nel parcheggio.
Nel parcheggio non possono esservi più auto con la medesima targa.

Consigli:

Consigliabile implementare un metodo ausiliario private int cercaTarga(String unaTarga) che cerca una targa nell'ArrayList box.

```
import java.util.ArrayList;
public class Autorimessa
{
 private ArrayList<String> box;
 /**Costruisce un oggetto della classe Autorimessa.*/
  public Autorimessa()
  {
     box=new ArrayList<String>();
  }
   /**Metodo ausiliario che cerca una targa nella lista box.
   @param unaTarga targa del veicolo da cercare.*/
    private int cercaTarga(String unaTarga)
    {
    int result=-1;
     for(int i=0;i<box.size();i++)
     {
      if(box.get(i).equals(unaTarga))
            result=i;
     }
     return result;
    }
```

```
/**Inserisce nella lista box una vettura.
   @param unaTarga targa del veicolo da inserire.*/
    public void parcheggia(String unaTarga)
    {
     if(cercaTarga(unaTarga)!=-1)
      return;
     box.add(unaTarga);
    }
    /**Elimina una vettura dalla lista box.
    @param unaTarga targa del veicolo da eliminare.*/
    public void rimuovi(String unaTarga)
    {
     if(cercaTarga(unaTarga)==-1)
      return;
    box.remove(cercaTarga(unaTarga));
    }
    /**Restituisce il numero di automobili presenti nella lista box.
    @return numero di veicoli presenti nel box.*/
    public int numeroAuto()
    {
     return box.size();
    }
}


import java.util.ArrayList;
public class AutorimessaTester
{
 public static void main(String[] args)
  {
   Autorimessa prova = new Autorimessa();
   prova.parcheggia("AY050JD");
   prova.parcheggia("AY050JD");
   prova.parcheggia("VY123FG");
   prova.parcheggia("SA100TA");
   prova.parcheggia("RA593IS");
   System.out.println(prova.numeroAuto());
   prova.rimuovi("AY050JD");
   System.out.println(prova.numeroAuto());
  }
}
```

Esercizio 7.7 - Ascensore

Testo:

Si desidera simulare un ascensore in funzione in un palazzo. L'ascensore ospita persone ed esegue le loro prenotazioni di spostamento ad un certo piano.
Costruire una classe Prenotazioni, con informazione su numero clienti prenotanti e numero piano prenotato. Sviluppare metodi di accesso ed un metodo toString che descriva la prenotazione stessa.
Costruire una classe Ascensore con le seguenti variabili di istanza: max numero piani, piano corrente, max numero persone, numero persone corrente, e lista prenotazioni. La lista delle prenotazioni ha una capienza massima prefissata, usare un array di Prenotazioni. Le prenotazioni vengono servite con la politica primo arrivato primo servito. Sviluppare i seguenti metodi. entra: incrementa il numero di persone nell'ascensore e mette in coda la relativa prenotazione. muovi: porta l'ascensore al piano specificato dalla prima prenotazione trovata, fa uscire le persone relative ed aggiorna la lista delle prenotazioni. toString: restituisce una stringa con la descrizione dello stato attuale dell'ascensore. In tutti i casi in cui venga violata una condizione (troppe persone in ascensore, troppe prenotazioni, ecc.) stampare un messaggio di errore ed uscire dal metodo relativo.

Consigli:
La classe Prenotazioni non presenta difficoltà particolari. Può risultare di più difficile risoluzione la classe Ascensore, in particolare il metodo muovi(), il quale presuppone di: individuare il piano cercato, sottrarre le persone che sono uscite dall'ascensore alla "fermata" richiesta ed, infine, aggiornare l'array delle prenotazioni.

```
public class Prenotazioni
{
private int persone;
private int piano;
/**Creo un oggetto della classe Prenotazioni.
@param nPersone Numero di persone che prenota.
@param nPiano Numero di piano.*/
 public Prenotazioni(int nPersone,int nPiano)
 {
  persone = nPersone;
  piano = nPiano;
 }
/**Restituisce il numero di persone.
@return Numero di persone.*/
 public int getPersone()
 {
  return persone;
 }
/**Restituisce il numero del piano.
@return Numero del piano.*/
 public int getPiano()
 {
  return piano;
 }
/**Stampa la situazione delle prenotazioni.
@return Situazione prenotazioni.*/
 public String toString()
 {
  String s="";
  s="il piano "+piano+" è stato prenotato da "+persone+" persone."+"\n";
  return s;
 }
}


import java.util.Scanner;
public class PrenotazioniTester
{
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);
  System.out.println("Inserire numero di piano");
  int nPiano = input.nextInt();
  System.out.println("Inserire numero di persone che desiderano recarsi a quel
piano");
  int nPersone = input.nextInt();
  Prenotazioni tester=new Prenotazioni(nPersone,nPiano);
  System.out.println(tester.toString());
 }
}
```

```
public class Ascensore
{
private final int MAXPIANI = 4;
private final int MAXPERSONE = 5;
private Prenotazioni[] prenota;
private int numPrenotazioni;
private int pianoCorr;
private int personeCorr;
/**Creo un oggetto della classe Ascensore.
@param unMax Il numero massimo di prenotazioni che si possono accodare.*/
 public Ascensore(int unMax)
 {
  prenota = new Prenotazioni[unMax];
  numPrenotazioni = 0;
  pianoCorr = 0;
  personeCorr = 0;
 }
/**Si incrementa il numero di persone nell'ascensore e
si mette in coda la relativa prenotazione quando ciò è possibile.
@param persone Le persone che vogliono entrare in ascensore.
@param piano Il piano cui queste persone vogliono andare.*/
 public boolean entra(int persone,int piano)
 {
  if (personeCorr + persone > MAXPERSONE)
    return false;
  else if (piano > MAXPIANI || piano < 0)
    return false;
  else if (numPrenotazioni == prenota.length)
     return false;
  else
 {
  personeCorr += persone;
  prenota[numPrenotazioni] = new Prenotazioni(persone,piano);
  numPrenotazioni++;
  return true;
 }
}
/**Porta l'ascensore al piano specificato dalla prima prenotazione
trovata, fa uscire le persone relative ed aggiorna la lista delle
prenotazioni.*/
 public void muovi()
 {
  if (numPrenotazioni > 0)
  {
   pianoCorr = prenota[0].getPiano();
   personeCorr = personeCorr - prenota[0].getPersone();
   System.arraycopy(prenota,1,prenota,0,prenota.length-1);
   numPrenotazioni --;
  }
 }
/**Stampa la descrizione dello stato attuale dell'ascensore,
 comprese prenotazioni in coda.
@return Lo stato attuale dell'ascensore.*/
 public String toString()
 {
  String r = "";
  for (int i = 0; i < numPrenotazioni; i++)
  {
   r = r + prenota[i].toString() + "\n";
   String s = "";
   return s = "Piano corrente: " + pianoCorr + "." + "\n" + "Persone in
ascensore:"”+ personeCorr + "\n" + r;
  }}
```

```
import java.util.Scanner;
public class AscensoreTester
{
 public static void main(String [] args)
 {
  Scanner in = new Scanner(System.in);
  System.out.println("Inserire il massimo delle prenotazioni possibili:");
  int max = in.nextInt();
  Ascensore elevator = new Ascensore(max);
  String t = "";
  int contatore = 0;
  while(!t.equalsIgnoreCase("q"))
  {
   System.out.println("inserire numero persone che effettuano una
prenotazione:");
   int n = in.nextInt();
   System.out.println("inserire il numero di piano per cui le "+n+" persone
hanno  prenotato:");
   int p = in.nextInt();
   System.out.println("L'ascensore arriva.");
   System.out.println(""+n+" persone salgono in ascensore.");
   elevator.entra(n,p);
   contatore ++;
   System.out.println("Info: per termirare le prenotazioni, digitare Q,
altrimenti  una lettera qualsiasi.");
   t=in.next();
   System.out.println("L'ascensore parte.");
   System.out.println(elevator.toString());
  }
  for (int i=0; i < contatore; i++)
  {
   elevator.muovi();
   System.out.println(elevator.toString());
  }
 }
}
```

Esercizio 7.8 - Biblioteca

Testo:

Una biblioteca desidera gestire elettronicamente il proprio archivio. Per fare cio` si implementi una classe Utente con le due variabili d'istanza String nome e String cognome, un costruttore Utente(String unNome, String unCognome) ed i metodi String getNome(), String getCognome() ed infine un metodo String toString() che restituisce una stringa con una descrizione dell'oggetto implicito, cioe` con i valori delle variabili d'istanza nome e cognome.
Si implementi inoltre una classe Libro con le tre variabili d'istanza int codice, String titolo ed Utente utenteAssegnato. La variabile utenteAssegnato e` di classe Utente e dovra` contenere un riferimento ad un oggetto di classe Utente che ha in prestito il libro. Nel caso il libro non sia in prestito, la variabile utenteAssegnato deve valere null. Implementare per la classe Libro un costruttore Libro(int unCodice, String unTitolo), il quale pone la variabile utenteAssegnato a null, ed i metodi Utente getUtenteAssegnato(), void setUtenteAssegnato(Utente unUtente), int getCodice(), String getTitolo() e String toString().
Implementare infine una classe Biblioteca avente le due variabili d'istanza ArrayList libri e ArrayList utenti. La variabile libri contiene tutti i libri della biblioteca. La variabile utenti contiene tutti gli utenti della biblioteca. Ricordate che ciascun elemento della lista libri deve avere la propria variabile d'istanza utenteAssegnato con il valore null oppure, nel caso il libro sia stato assegnato ad un utente, tale variabile conterra` un riferimento ad un utente presente nella lista utenti.

Sviluppare i seguenti metodi per la classe Biblioteca. Un costruttore Biblioteca(). La biblioteca e` inizialmente vuota, cioe` non contiene alcun libro ne' alcun utente. Un metodo void aggiungiUtente(String unNome, String unCognome) che crea un nuovo utente e lo aggiunge alla lista utenti. Un metodo void aggiungiLibro(int unCodice, String unTitolo) che crea un nuovo libro e lo aggiunge alla lista libri. Per tale libro il valore della variabile d'istanza utenteAssegnato dovra` essere null, poiche` il libro non e` ancora in prestito. Un metodo void creaPrestito(int unCodice, String unCognome). Tale metodo cerchera` nella lista libri un elemento avente il codice unCodice e cerchera` nella lista utenti un utente avente il cognome unCognome. La variabile d'istanza utenteAssegnato del libro trovato dovra` infine assumere come valore un riferimento all'utente trovato, in modo da registrare il prestito. Infine, sviluppare un metodo String toString() per visualizzare in forma di stringa lo stato della biblioteca (tutti i suoi libri ed utenti).

Consigli:
Si assuma che non esistano piu` libri con lo stesso codice, e che non esistano piu` utenti con lo stesso cognome. Le classi Utente e Libro non presentano particolari difficoltà. Nella classe biblioteca il metodo creaPrestito(int unCodice, String unCognome) per implementarlo e` necessario svolgere due operazioni di ricerca, puo` essere d'aiuto in questo caso usare due metodi ausiliari per svolgere tali ricerche.

```
public class Utente
{
private String nome;
private String cognome;
/**Crea un oggetto della classe Utente.
 @param unNome Nome.
 @param unCognome Cognome.*/
 public Utente(String unNome, String unCognome)
 {
  nome = unNome;
  cognome = unCognome;
 }
/**Restituisce il nome dell'utente.
 @return Nome.*/
 public String getNome()
 {
  return nome;
 }
/**Restituisce il cognome dell'utente.
 @return Cognome.*/
 public String getCognome()
 {
  return cognome;
 }
/**Restituisce nome e cognome dell'utente.
 @return Nome e cognome dell'utente.*/
 public String toString()
 {
  return nome + " " + cognome;
 }
}
```

```
public class Libro
{
private int codice;
private String titolo;
private Utente utenteAssegnato;
```

```
/**Crea un oggetto della classe Libro.*/
 public Libro(int unCodice, String unTitolo)
 {
  codice = unCodice;
  titolo = unTitolo;
  utenteAssegnato = null;
 }
/**Restituisce l'utente assegnato.
 @return L'utente assegnato.*/
 public Utente getUtenteAssegnato()
 {
  return utenteAssegnato;
 }
/**Assegna un utente.
 @param unUtente Un oggetto della classe Utente.*/
 public void setUtenteAssegnato(Utente unUtente)
 {
  utenteAssegnato = unUtente;
 }
/**Restituisce la condizione dell'oggetto libro.
 @return Stituazione oggetto.*/
 public String toString()
 {
  return utenteAssegnato + " " +codice+ " "+titolo;
 }
/**Restituisce il codice.
 @return Codice.*/
 public int getCodice()
 {
  return codice;
 }
}



import java.util.ArrayList;
public class Biblioteca
{
private ArrayList<Libro> libri;
private ArrayList<Utente> utenti;

/**Costruisce un oggetto della classe Biblioteca.*/
 public Biblioteca()
 {
  utenti = new ArrayList<Utente>();
  libri = new ArrayList<Libro>();
 }
/**Metodi Ausiliari.
 Cerca un codice.
 @param unCodice Codice da cercare.
 @return Indice del codice.*/
 private int cercaCodice(int unCodice)
 {
 int result=-1;
 for(int i=0;i<libri.size();i++)
 {
  if(libri.get(i).getCodice()==unCodice)
    result=i;
 }
 return result;
}
```

```
/**Cerca un utente.
 @param unCognome Cognome dell'utente da cercare.
 @return Indice dell'utente.*/
 private int cercaUtente(String unCognome)
 {
  int result=-1;
  for(int i=0;i<utenti.size();i++)
  {
   if(utenti.get(i).getCognome().equals(unCognome))
     result=i;
  }
  return result;
 }
/**Metodi richiesti.
 Aggiunge un utente alla biblioteca.
 @param unNome Nome.
 @param unCognome Cognome.*/
 public void aggiungiUtente(String unNome, String unCognome)
 {
  if(cercaUtente(unCognome)!=-1)
    return;
  Utente a = new Utente(unNome, unCognome);
  utenti.add(a);
 }
/**Aggiunge un libro alla biblioteca.
 @param unCodice Codice.
 @param unTitolo Titolo.*/
 public void aggiungiLibro(int unCodice, String unTitolo)
 {
  if(cercaCodice(unCodice)!=-1)
    return;
  Libro b = new Libro(unCodice, unTitolo);
  libri.add(b);
 }
/**Crea un prestito, assegnando un libro a un utente.
 @param unCodice Codice.
 @unCognome Cognome.*/
 public void creaPrestito(int unCodice, String unCognome)
 {
libri.get(cercaCodice(unCodice)).setUtenteAssegnato(utenti.get(cercaUtente(unCog
nome)));
 }
/**Restituisce il contenuto degli ArrayList.
 @return Situazione biblioteca.*/
 public String toString()
 {
  String r = "";
  r = "Libri="+libri+" Utenti="+utenti;
  return r;
 }
}



import java.util.Scanner;
public class BibliotecaTester
{
 public static void main(String[] args)
 {
  Scanner in = new Scanner(System.in);
  Biblioteca genna = new Biblioteca();
  boolean d = false;
  while (!d)
```

```
  {
   System.out.println("Scegliere l'operazione da effettuare:");
   System.out.println("1) aggiungi un utente");
   System.out.println("2) aggiungi un libro");
   System.out.println("3) crea un prestito");
   System.out.println("4) stampa il contenuto della biblioteca");
   System.out.println("5) esci");
   int operazione = in.nextInt();
   if (operazione == 1)
   {
    System.out.println("Scegliere il nome utente:");
    String a= in.nextLine();
    String nome = in.nextLine();
    System.out.println("Scegliere il cognome utente:");
    String cognome = in.nextLine();
    genna.aggiungiUtente(nome,cognome);
   }
   else if (operazione == 2)
   {
    System.out.println("Scegliere il titolo del libro:");
    String a= in.nextLine();
    String titolo = in.nextLine();
    System.out.println("Scegliere il codice del libro:");
    int codice = in.nextInt();
    genna.aggiungiLibro(codice,titolo);
   }
   else if (operazione == 3)
   {
    System.out.println("Scegliere il cognome dell'utente che richiede il
prestito:");
    String a= in.nextLine();
    String cognome = in.nextLine();
    System.out.println("Scegliere il codice del libro da prestare:");
    int codice = in.nextInt();
    genna.creaPrestito(codice,cognome);
   }
   else if (operazione == 4)
   {
    System.out.println("Il contenuto della biblioteca:");
    System.out.println(genna.toString());
   }
   else if (operazione == 5)
   {
   d = true;
   }
  }
 }
}
```

Esercizio 7.9 Gestione di un'assicurazione

Testo:

Un'assicurazione desidera creare un archivio elettronico in grado di raccogliere informazioni sulle automobili e sui loro proprietari. Si implementi una classe Cliente, avente il nominativo (stringa) come variabile d'istanza; una classe Automobile avente come variabili d'istanza il numero di targa della vettura (intero) e un riferimento al proprietario della classe Cliente. La classe Archivio presenta due variabili di istanza ArrayList<Cliente> clienti e ArrayList<Automobile> automobili. Dotare la classe Archivio dei seguenti metodi: addCliente(String unNome)che aggiunge un nuovo cliente all'arraylist clienti. addAutomobile(int unaTarga, String unNome) che aggiunge una nuova auto all'arraylist automobili. removeAutomobile(int unaTarga) che rimuove un automobile dall'arraylist automobili. clienteTarga(int unaTarga) che restituisce

il nome del proprietario di una targa. returnTarga(String unNome) che Restituisce una lista con le targhe delle automobili aventi un dato proprietario. contaAuto(String unNome) che conta quante automobili sono associate a un cliente. removeCliente(String unNome) che toglie un cliente dall'arraylist clienti e le sue auto dall'arraylist automobili. maxAuto() che restituisce il cliente con il maggior numero di auto.

Consigli:
I problemi maggiori possono essere riscontrati nella classe Archivio: si consiglia l'uso di metodi ausiliari. Si prega di prestare attenzione alla stesura del metodo removeCliente(String unNome) dove è richiesto di cancellare una serie di elementi(che possono essere contigui) da un Arraylist. È quindi necessario adottare alcuni accorgimenti(Esercizio 7.1 cancArrayList).

```
public class Cliente
{
private String nome;
/**Crea un oggetto della classe Cliente.
 @param unNome Nome.*/
 public Cliente(String unNome)
 {
  nome=unNome;
 }
/**Restituisce il nome del cliente.
 @return Nome del cliente.*/
 public String getNome()
 {
  return nome;
 }
}
```

```
public class Automobile
{
private int targa;
private Cliente proprietario;
/**Crea un oggetto della classe Automobile.
 @param unaTarga Targa automobile.
 @param unCliente Oggetto della classe Cliente.*/
 public Automobile (int unaTarga, Cliente unCliente)
 {
  targa = unaTarga;
  proprietario = unCliente;
 }
/**Restituisce la targa di un auto.
 @return Targa.*/
 public int getTarga()
 {
  return targa;
 }
/**Restituisce il proprietario di un auto.
 @return Oggetto della classe Cliente.*/
 public Cliente getProprietario()
 {
  return proprietario;
 }
}
```

```
import java.util.ArrayList;
public class Archivio
{
private ArrayList<Cliente> clienti;
private ArrayList<Automobile> automobili;
/**Crea un oggetto della classe Archivio.*/
 public Archivio()
 {
  clienti = new ArrayList<Cliente>();
  automobili = new ArrayList<Automobile>();
 }
/**Metodi ausiliari.
Restituisce l'indice del cliente nell'arraylist clienti.
@param unNome Nome.
@return Indice.*/
 private int indiceCliente(String unNome)
 {
  int result=-1;
  for(int i = 0;i<clienti.size();i++)
  {
   if(clienti.get(i).getNome().equalsIgnoreCase(unNome))
     result=i;
  }
  return result;
 }
/**Restituisce l'indice della targa nell'arraylist automobili.
 @param unaTarga Targa.
 @return Indice.*/
 private int indiceTarga(int unaTarga)
 {
  int result=-1;
  for(int i=0;i<automobili.size();i++)
  {
   if (automobili.get(i).getTarga() == unaTarga)
     result=i;
  }
  return result;
 }
/**Metodi richiesti.
 Aggiunge un nuovo cliente all'arraylist clienti.
 @param unNome Nome.*/
 public void addCliente(String unNome)
 {
  if(indiceCliente(unNome)!= -1)
    return;
  clienti.add(new Cliente(unNome));
 }
/**Aggiunge una nuova auto all'arraylist automobili.
 @param unTarga Targa.
 @param unNome Nome.*/
 public void addAutomobile(int unaTarga, String unNome)
 {
  if(indiceTarga(unaTarga)!= -1)
    return;
  addCliente(unNome);
  automobili.add(new Automobile(unaTarga,clienti.get(indiceCliente(unNome))));
 }
```

```
/**Rimuove un automobile dall'arraylist automobili.
 @param unaTarga.*/
 public void removeAutomobile(int unaTarga)
 {
  if(indiceTarga(unaTarga)==-1)
    return;
  automobili.remove(indiceTarga(unaTarga));
 }
/**Restituisce il nome del proprietario di una targa.
 @param unaTarga Targa.*/
 public String clienteTarga(int unaTarga)
 {
  if(indiceTarga(unaTarga)== -1)
    return "Targa non presente.";
  else
    return automobili.get(indiceTarga(unaTarga)).getProprietario().getNome();
 }
/**Restituisce una lista con le targhe delle automobili aventi un dato
proprietario.
 @param unNome Nome.
 @return Lista con le targhe.*/
 public ArrayList<Integer> returnTarga(String unNome)
 {
  ArrayList<Integer> temp = new ArrayList<Integer>();
  if(indiceCliente(unNome)==-1)
    return temp;
  for(int i=0;i<automobili.size();i++)
  {
   if(automobili.get(i).getProprietario().getNome().equals(unNome))
     temp.add(automobili.get(i).getTarga());
  }
  return temp;
 }
/**Conta quante automobili sono associate a un cliente.
 @param unNome Nome.
 @return Numero automobili associate ad un cliente.*/
 public int contaAuto(String unNome)
 {
  return returnTarga(unNome).size();
 }
/**Toglie un cliente dall'arraylist clienti e le sue auto dall'arraylist
automobili.
 @param unNome Nome.*/
 public void removeCliente(String unNome)
 {
  if(indiceCliente(unNome)== -1)
    return;
  for(int i=automobili.size()-1;i>-1;i--)
  {
   if(automobili.get(i).getProprietario().getNome().equals(unNome))
     automobili.remove(i);
  }
  clienti.remove(indiceCliente(unNome));
 }
/**Restituisce il cliente con il maggior numero di auto.
 @return Cliente.*/
```

```
public Cliente maxAuto()
 {
  int indice=-1;
  int max=0;
  for(int i=0;i<clienti.size();i++)
  {
   int proprietario=0;
   proprietario=contaAuto(clienti.get(i).getNome());
   if(proprietario>max)
   {
    max=proprietario;
    indice=i;
   }
  }
  return clienti.get(indice);
 }
}
```

```
public class ArchivioTester
{
 public static void main(String args[])
 {
  Archivio alianz = new Archivio();
  alianz.addCliente("Rossi");
  alianz.addCliente("Pizzato");
  alianz.addCliente("Bressan");
  alianz.addAutomobile(2534,"Pizzato");
  alianz.addAutomobile(6484,"Borgato");
  alianz.addAutomobile(8960,"Rossi");
  alianz.addAutomobile(8743,"Bressan");
  alianz.addAutomobile(2539,"Pizzato");
  System.out.println(alianz.returnTarga("Rossi"));
  System.out.println(alianz.clienteTarga(6484));
  System.out.println(alianz.contaAuto("Bressan"));
  System.out.println(alianz.maxAuto());
  alianz.removeCliente("Bressan");
  System.out.println(alianz.contaAuto("Bressan"));
 }
}
```

Esercizio 7.10 - Gestione di un campo da tennis

Testo:

Fare una classe Prenotazione (di un campo da tennis) contenente il nome del cliente e l'ora della sua prenotazione. Implementare una classe Campo in cui ci sono i seguenti metodi: public boolean addPren(int inizio, int fine, String unNomeCliente), per prenotare il campo (controlla se i dati inseriti sono giusti e se il campo è disponibile dopodichè salva la prenotazione e restituisce true se il campo è stato prenotato); public boolean removePren (int inizio, int fine, String unNomeCliente), per cancellare una prenotazione (controlla se il campo è stato prenotato dal cliente che vuole cancellare la prenotazione dopodichè la cancella e restituisce true se la prenotazione è stata cancellata); public String toString(); public double utilizzo(), per trovare la percentuale dell'utilizzo del campo.

Consigli:

Nello sviluppo di alcuni metodi, si consiglia il seguente procedimento:

- addPren (int inizio, int fine, String unNomeCliente): creare un metodo ausiliario che individua se il campo da gioco è disponibile.
- removePren (int inizio, int fine, String unNomeCliente): anche in questo caso s'invita ad utilizzare un metodo ausiliario per verificare se esiste la prenotazione da disdire.

```
public class Prenotazione
{
private int ora;
private String nomeCliente;
/**Crea un oggetto della classe Prenotazione.
 @param unOra Orario.
 @param unCliente Cliente.*/
 public Prenotazione(int unOra, String unCliente)
 {
  ora=unOra;
  nomeCliente=unCliente;
 }
/**Restituisce l'ora.
 @return Ora.*/
 public int getOra()
 {
  return ora;
 }
/**Restituisce il cliente.
 @return Cliente.*/
 public String getCliente()
 {
  return nomeCliente;
 }
/**Restituisce la prenotazione.
 @return Prenotazione.*/
 public String toString()
 {
  return "Ore: "+ora+" campo prenotato dal Sig. "+nomeCliente;
 }
}
```

```
public class CampoTennis
{
public final int APERTURA = 9;
public final int CHIUSURA = 21;
private Prenotazione[] prenotazioni;
/**Crea un oggetto della classe CampoTennis.*/
 public CampoTennis()
 {
  prenotazioni = new Prenotazione[CHIUSURA-APERTURA];
 }
/**Metodi ausiliari.
 Controlla se il campo è disponibile in un dato intervallo.
 @param inizio Inizio.
 @param fine Fine.
 @return True se è libero.*/
```

```
private boolean isDisp(int inizio, int fine)
 {
  boolean result = true;
  for(int i = inizio - APERTURA; (i < fine - APERTURA)&&(result); i++)
  {
   if(prenotazioni[i] != null)
     result = false;
  }
  return result;}
/**Controlla se il campo è prenotato in un dato intervallo da un dato cliente.
 @param inizio Inizio.
 @param fine Fine.
 @param unNomeCliente.
 @return True se è libero.*/
 private boolean isPren (int inizio, int fine, String unNomeCliente)
 {
  boolean result = true;
  for(int i = inizio - APERTURA; (i < fine - APERTURA)&&(result); i++)
  {
  if(prenotazioni[i]==null||!prenotazioni[i].getCliente().equals(unNomeCliente))
   result = false;
  }
  return result;
 }
/**Metodi richiesti.
  Prenota il campo.
  @return true se la prenotazione è andata a buon fine.*/
 public boolean addPren(int inizio, int fine, String unNomeCliente)
 {
 boolean result = false;
 if(fine - inizio>0&&inizio >= APERTURA&&fine <= CHIUSURA)
   if(isDisp(inizio,fine))
   {
    for(int i = inizio - APERTURA; i < fine - APERTURA; i++)
    {
     prenotazioni[i] = new Prenotazione(APERTURA + i,unNomeCliente);
        result = true;
    }
   }
   return result;
 }
/**Cancella una prenotazione.
  @param inizio Inizio.
  @param fine Fine.
  @param unNomeCliente.
  @return True se la cancellazione va buon fine.*/
 public boolean removePren (int inizio, int fine, String unNomeCliente)
 {
 boolean result = false;
 if(fine - inizio > 0 && inizio >= APERTURA && fine <= CHIUSURA)
   if(isPren(inizio, fine, unNomeCliente))
   {
    for(int i = inizio - APERTURA; i < fine - APERTURA; i++)
    {
     prenotazioni[i] = null;
     result = true;
    }
   }
   return result;
 }
/**Restituisce la situazione del campo.
 @return Situazione del campo.*/
```

```
public String toString()
 {
  String temp="";
  for(int i = 0; i < prenotazioni.length; i++)
  {
   if(prenotazioni[i] == null)
     temp = temp+"Ore: "+(i+APERTURA)+" campo libero"+"\n";
   else
     temp = temp + prenotazioni[i].toString() + "\n";
  }
  return temp;
 }
/**Trova la percentuale dell'utilizzo del campo.
 @return Percentuale.*/
 public double utilizzo()
 {
  int conta = 0;
  for (int i = 0; i < CHIUSURA - APERTURA; i++)
  {
   if(prenotazioni[i] != null)
     conta++;
  }
  return conta * 10000 / (CHIUSURA - APERTURA) / 100.0;
 }
}


public class CampoTennisTester
{
 public static void main(String[] args)
 {
  CampoTennis campo1 = new CampoTennis();
  System.out.println(campo1.addPren(9,10,"Rossi"));
  System.out.println(campo1.addPren(10,12,"Ziorzi"));
  System.out.println(campo1.addPren(11,18,"Parolin"));
  System.out.println(campo1.addPren(14,19,"Parolin"));
  System.out.println(campo1.toString());
  System.out.println(campo1.utilizzo());
  System.out.println(campo1.removePren(16,17,"Parolin"));
  System.out.println(campo1.addPren(19,21,"Timelli"));
  System.out.println(campo1.toString());
  System.out.println(campo1.utilizzo());
 }
}
```

Esercizio 7.11 Gestione di una compagnia aerea

Testo:

Una compagnia aerea desidera gestire elettronicamente le prenotazioni sui singoli voli. Abbiamo una classe Cliente, con nome unico (si assuma che non ci siano clienti omonimi). Poi una classe Volo, che contiene due variabili d'istanza denominate posti e attesa. Ogni volo ha un numero max di passeggeri (passato come parametro al costruttore). Quando un cliente prenota il volo, viene inserito in posti se c'e` ancora disponibilita`, altrimenti viene inserito in attesa. Quando un cliente in posti disdice la sua prenotazione, si libera un posto e prendiamo il primo elemento di attese e lo trasferiamo su posti. Notate che l'ordine degli elementi nell'array posti e' irrilevante, mentre l'ordine degli elementi nell'ArrayList attese e' rilevante. Attesa e` un ArrayList senza limitazioni sulla dimensione. Per l'array posti, usate la tecnica della "sentinella" che deve essere anch'essa una variabile d'istanza.

Consigli:
Posti deve essere gestito come un array parzialmente riempito, utilizzando cioè la variabile sentinella (= prossimo posto libero) numero, quindi solo la parte superiore dell'array viene utilizzata, e non esistono 'buchi'.
È consigliato implementare due metodi ausiliari che controllino la presenza di un cliente in posti e in attesa.
L'unico problema, invece, che riguarda il metodo statistica(char ch) riguarda l'introduzione del tipo char: per facilitarne la comprensione, s'invita a rivedere il capitolo 4. Esso, comunque, può esser utile per individuare un carattere specifico di una stringa.

```
public class Cliente
{
private String cognome;
/**Crea un oggetto della classe Cliente.
 @param unCognome Cognome.*/
 public Cliente(String unCognome)
 {
  cognome = unCognome;
 }
/**Restituisce il cognome.
 @return Cognome.*/
 public String getCognome()
 {
  return cognome;
 }
}
```

```
import java.util.Arrays;
import java.util.ArrayList;
public class Volo
{
private int numero;
private Cliente[] posti;
private ArrayList<Cliente> attesa;
/**Crea unoggetto della classe Volo.
 @param numPass Numero massimo di passeggeri.*/
 public Volo (int numPass)
 {
  numero = 0 ;
  posti = new Cliente[numPass];
  attesa = new ArrayList<Cliente>();
 }
/**Metosi ausiliari.
 Trova cliente in array posti.
 @param unCognome Cognome.
 @return indice.*/
 private int clientePosti(String unCognome)
 {
  int result=-1;
  for(int i=0;i<numero;i++)
  {
   if(posti[i].getCognome().equals(unCognome))
     result=i;
  }
  return result;
 }
/**Trova cliente in ArrayList attesa.
 @param unCognome Cognome.
 @return indice.*/
```

```
 private int clienteAttesa(String unCognome)
 {
  int result=-1;
  for(int i=0;i<attesa.size();i++)
  {
   if(attesa.get(i).getCognome().equals(unCognome))
     result=i;
  }
  return result;
 }
/**Inserisce una prenotazione.
@param unCognome Cognome.
@return True se la prenotazione va a buon fine false se viene messo in attesa.*/
 public boolean prenota(String unCognome)
 {
  boolean result=true;
  if(clientePosti(unCognome)!=-1)
    return false;
  Cliente unCliente = new Cliente(unCognome);
  if(numero<posti.length)
  {
   posti[numero]=unCliente;
   numero++;
  }
  else
  {
   attesa.add(unCliente);
   result=false;
  }
 return result;
 }
/**Elimina una prenotazione, il primo cliente in attesa viene inserito
nell'array posti.
 @param unCognome Cognome.*/
 public void disdici(String unCognome)
 {
  if(clientePosti(unCognome)!=-1)
  {
   int indice=clientePosti(unCognome);
   System.arraycopy(posti,indice+1,posti,indice,posti.length-indice-1);
   if(attesa.isEmpty()==false)
   {
    posti[numero-1] = attesa.get(0);
    attesa.remove(0);
   }
   else
     numero --;
  }
  if(clientePosti(unCognome)==-1)
  {
   if(clienteAttesa(unCognome)!=-1)
     attesa.remove(clienteAttesa(unCognome));
   else
     return;
  }
 }
/**Restituisce la situazione della classe volo.
 @return Situazione.*/
 public String toString()
 {
  return "Sono prenotati per il volo "+Arrays.toString(posti)+
   " clienti.\nSono in lista d'attesa "+attesa.size()+" clienti.";
 }
```

```
/**Restituisce il numero di clienti aventi cognome che inizia con la lettera
data.
 @param ch Lettera iniziale.
 @return Numero di clienti.*/
 public int statistica(char ch)
 {
  int cont = 0;
  for(int i = 0;i < numero; i++)
  {
   if(posti[i].getCognome().charAt(0) == ch)
     cont++;
  }
  for(int j=0;j< attesa.size();j++)
  {
   if(attesa.get(j).getCognome().charAt(0) == ch)
     cont++;
  }
  return cont;
 }
}



public class VoloTester
{
 public static void main(String[] args)
 {
  Volo qatarAirways=new Volo(5);
  qatarAirways.prenota("Pizzato");
  qatarAirways.prenota("Rossi");
  qatarAirways.prenota("Rigoni");
  qatarAirways.prenota("Zorzi");
  qatarAirways.prenota("Fioretto");
  qatarAirways.prenota("Bressan");
  System.out.println(qatarAirways.statistica('R'));
  System.out.println(qatarAirways.toString());
  qatarAirways.disdici("Zorzi");
  qatarAirways.disdici("Rigoni");
  qatarAirways.prenota("Ronchi");
  System.out.println(qatarAirways.toString());
 }
}
```

CAPITOLO 8 Progettazione di Classi

REMIND:

(Paragrafo 8.4) EFFETTI COLLATERALI

- *effetto collaterale* → qualunque tipo di comportamento al di fuori del metodo stesso.

(Paragrafo 8.4) AMBITO DI VISIBILITÀ

- *ambito di visibilità* → porzione di programma all'interno della quale si può accedere alla variabile.

  Ambito di visibilità di:
  - *variabile locale* → dal punto in cui viene creata fino alla fine del blocco ({…}) che la contiene.
  - *parametro esplicito* → dall'inizio alla fine del metodo che lo contiene.
  - *variabili d'istanza* → visibilità di classe (in tutta la classe).

- Se si utilizza una variabile d'istanza nel campo i visibilità di una variabile locale con lo stesso nome, prevale la variabile locale, che mette in ombra il campo d'istanza. Per riferirsi alla variabile d'istanza si può utilizzare la forma this.variabileDiIstanza

Nota: in questo capitolo non sono preseti esercizi significativi, ragion per cui non vengono inseriti in questa dispensa.

TEMI D'ESAME SVOLTI (Esercizi 5)

LettoreVideo-25/01/10

Testo:

Una società che produce apparecchi multimediali desidera sviluppare il software per la gestione di un lettore video portatile. Si consideri la classe Genere, avente come variabile d'istanza un nome (stringa) ed un numero di visualizzazioni (intero). Si consideri inoltre la classe Video, avente come variabili d'istanza un titolo (stringa) ed un riferimento ad un oggetto di classe Genere. Si assuma siano già disponibili gli usuali metodi costruttori, accessori e modificatori per ciascuna delle due classi. Sviluppare la seguente classe:

```
public class LettoreVideo {
private ArrayList<Video> video;
private Genere[] generi;
private int numGeneri;
private final int MAX_GENERI = 35;
public LettoreVideo(){...}
public void aggGenere(String unNome){...}
public void aggVideo(String unTitolo, String unGenere){...}
public void playVideo(String unTitolo){...}
public int contaVideo(String unGenere){...}
public void rimuoviGenere(String unGenere){...}
public ArrayList<Genere> top(int soglia) {...}
}
```

Le variabili d'istanza video e generi contengono rispettivamente tutti i video ed i generi presenti nel
lettore. La variabile generi è un array di dimensione MAX GENERI, e deve essere gestito come un array parzialmente riempito. Il metodo aggVideo inserisce nel lettore un nuovo video (se non esiste già) con un riferimento all'oggetto di classe Genere appropriato nella arraylist generi. Se il genere non esiste già, deve essere creato un oggetto che lo rappresenti. Il metodo playVideo incrementa di una unità il numero di visioni del genere associato al video specificato. Il metodo contaVideo restituisce il numero di video presenti nel lettore che sono stati classificati con il genere specificato. Il metodorimuoviGenere rimuove dal lettore il genere specificato assieme a tutti i video classificati con tale genere. Il metodo top restituisce una lista con tutti i generi che hanno totalizzato un numero di visioni pari o superiore alla soglia indicata.

Consiglio:

In questo esercizio la difficoltà è gestire un array parzialmente riempito, cosa che viene spesso richiesta negli esercizi d'esame. Per questo il presente esercizio verrà ampliamente commentato al fine di renderlo il più chiaro possibile.

La classe Genere non presenta alcuna difficoltà

```
public class Genere
{
private String nome;
private int numero;
```

```
/**Creo un oggetto della classe Genere.
 @param unNome Nome del genere.*/
 public Genere(String unNome)
 {
  nome = unNome;
  numero = 0;
 }
/**Restituisce il nome del genere.
 @return Nome del genere.*/
 public String getNome()
 {
  return nome;
 }
/**Restituisce il numero del genere.
 @return Numero del genere.*/
 public int getNumero()
 {
  return numero;
 }
/**Incrementa di una unità il numero del genere.*/
 public void aggiungi()
 {
  numero++;
 }
}
```

La classe Video non presenta alcuna difficoltà

```
public class Video
{
private String titolo;
private Genere genere;
/**Creo un oggetto della classe Video.
 @param unTitolo Titolo del video.
 @param unGenere Oggetto della classe Genere.*/
 public Video(String unTitolo, Genere unGenere)
 {
  titolo = unTitolo;
  genere = unGenere;
 }
/**Restituisce il titolo del video.
 @return Titolo.*/
 public String getTitolo()
 {
  return titolo;
 }
/**Restituisce il genere del video.
 @return Genere.*/
 public Genere getGenere()
 {
  return genere;
 }
}
```

La classe LettoreVideo presenta varie difficoltà

```
import java.util.ArrayList;
public class LettoreVideo
{
private ArrayList<Video>video;
private Genere[] generi;
private int numGeneri;
private final int MAX_GENERI=35;
```

Il costruttore deve inizializzare le variabili di istanza.

```
/**Creo un oggetto della classe video.*/
 public LettoreVideo()
 {
  video = new ArrayList<Video>();
  generi = new Genere[MAX_GENERI];
  numGeneri = 0;
 }
```

Vengono creati due metodi ausiliari per semplificare la stesura dei metodi richiesti, il primo metodo ausiliario cerca il genere nell'Array generi, restituisce l'indece in corrispondenza del quale si trova l'oggetto cercato(se presente), nel caso in cui l'oggetto non venga trovato restituisce -1.

```
/**Metodi ausiliari.
 Cerca il genere.
 @param unNome Nome del genere.
 @return Indice.*/
 private int cercaGenere(String unNome)
 {
  int result = -1;
  for(int i = 0;i < numGeneri; i++)
  {
   if(generi[i].getNome().equals(unNome))
     result = i;
  }
  return result;
 }
```

Il secondo metodo ausiliario cerca il video nell'ArrayList video, restituisce l'indece in corrispondenza del quale si trova l'oggetto cercato(se presente), nel caso in cui l'oggetto non venga trovato restituisce -1.

```
/**Cerca il video.
 @param unTitolo Titolo del video.
 @return Indice.*/
 private int cercaVideo(String unTitolo)
 {
  int result = -1;
  for(int i = 0;i < video.size(); i++)
  {
   if(video.get(i).getTitolo().equals(unTitolo))
     result = i;
  }
  return result;
 }
```

La prima istruzione controlla che il genere non sia già presente nell'Array generi(per controllare si usa il metodo ausiliario cercaGenere) e che l'Array generi non sia pieno. Se il genere è già presente esce dal metodo(return), altrimenti crea un oggetto della classe Genere e lo inserisce nell'Array generi nella posizione della "sentinella"(numGeneri) che successivamente viene abbassata di una posizione.

```
/**Metodi richiesti.
 Inserisce un genere in generi.
 @param unNome Nome del genere.*/
 public void aggGenere(String unNome)
 {
  if(cercaGenere(unNome)!=-1||numGeneri>=MAX_GENERI)
    return;
  Genere a = new Genere(unNome);
  generi[numGeneri] = a;
  numGeneri++;
 }
```

Le prime istruzioni verificano che il video e il genere non siano già presenti. Se il video è già presente nell'ArrayList video esce dal metodo(return). Se il video non è presente ma il genere è già presente crea il video e lo inserisce nell'ArrayList video. Se non sono presenti ne il genere ne il video crea entrambi(se l'Array generi non è pieno) e li inserisce nelle rispettive liste: il genere viene aggiunto richiamando il metodo aggGenere spiegato sopra, il video viene creato ed aggiunto all'ArrayList video.

```
/**Inserisce nel lettore un nuovo video con un riferimento all'oggetto di classe
Genere. Se il genere non esiste, deve essere creato.
@param unTitolo Titolo del video.
@param unGenere Nome del genere.*/
 public void aggVideo(String unTitolo, String unGenere)
 {
  if(cercaVideo(unTitolo)!=-1)
    return;
  if(cercaGenere(unGenere)!=-1)
  {
   Video a = new Video(unTitolo, generi[cercaGenere(unGenere)]);
   video.add(a);
  }
  if(cercaGenere(unGenere)==-1&&numGeneri<MAX_GENERI)
  {
   aggGenere(unGenere);
   Video b = new Video(unTitolo,generi[cercaGenere(unGenere)]);
   video.add(b);
  }
 }
```

La prima istruzione controlla (con il metodo ausiliario cercaVideo) che il video sia presente nell'ArrayList video, se non è presente esce dal metodo(return). L'ultima istruzione sfrutta il metodo aggiungi() della classe Genere per incrementare di una unità la variabile di istanza dell'oggetto di tipo Genere che è associato al video.

```
/**Incrementa di una unità il numero di visioni del genere associato al video.
 @param unTitolo Titolo del video.*/
 public void playVideo(String unTitolo)
 {
  if(cercaVideo(unTitolo)==-1)
    return;
  video.get(cercaVideo(unTitolo)).getGenere().aggiungi();
 }
```

La prima istruzione controlla (con il metodo ausiliario cercaGenere) che il genere sia presente nell'Array generi, se non è presente restituisce -1. L'istruzione successiva inizializza la variabile int num, necessaria per il calcolo, uguale a zero. Il ciclo for visita tutti gli oggetti presenti nell'ArrayList video ed estrae da essi il nome del genere, che viene confrontato con la stringa passata come parametro esplicito. Se le stringhe sono uguali la variabile num viene incrementata di una unità.

```
/**Restituisce il numero di video presenti con il genere specificato.
 @param unGenere Nome del genere.
 @return Numero di video.*/
 public int contaVideo(String unGenere)
 {
  if(cercaGenere(unGenere)==-1)
    return -1;
  int num = 0;
  for(int i = 0;i<video.size(); i++)
  {
   if(video.get(i).getGenere().getNome().equals(unGenere))
     num++;
  }
  return num;
 }
```

La prima istruzione controlla (con il metodo ausiliario cercaGenere) che il genere sia presente nell'Array generi, se non è presente esce dal metodo (return). Il ciclo for visita dall'ultimo al primo (poiché potrebbero esserci più elementi contigui da cancellare, si invita a rivedere l'esercizio 7.1)tutti gli oggetti presenti nell'ArrayList video ed estrae da essi il nome del genere che viene confrontato con la stringa passata come parametro esplicito. Se le stringhe sono uguali il video viene rimosso dall'ArrayList video. Le ultime istruzioni eliminano il genere (in questo caso l'ordine degli oggetti di tipo Genere nell'array generi non è rilevante per cui si prende l'ultimo elemento dell'array e lo si inserisce al posto di quello da eliminare) e spostano la sentinella verso l'alto di una posizione.

```
/**Rimuove dal lettore il genere specificato assieme a tutti i video
classificati con tale genere.
@param rimuoviGenere Nome del genere da rimuovere.*/
 public void rimuoviGenere(String rimuoviGenere)
 {
  if(cercaGenere(rimuoviGenere)==-1)
    return;
  for(int i=video.size()-1;i>-1;i--)
  {
   if(video.get(i).getGenere().getNome().equals(rimuoviGenere))
     video.remove(i);
  }
  generi[cercaGenere(rimuoviGenere)] = generi[numGeneri-1];
  numGeneri--;
 }
```

La prima istruzione inizializza un ArrayList top. Il ciclo for visita gli elementi(non necessariamente tutti, solo quelli compresi tra l'indice zero dell'Array e la "sentinella" numGeneri), ne estrae il numero di visualizzazioni che viene confrontato con la soglia(passata come parametro esplicito). Se il numero di visualizzazioni è maggiore o uguale alla soglia il genere viene aggiunto all'ArrayList top.

```
/**Restituisce un ArrayList con tutti i generi che hanno totalizzato un numero
di visioni pari o superiore alla soglia indicata.
@param soglia Soglia.
@return ArrayList top.*/
 public ArrayList<Genere> top(int soglia)
 {
  ArrayList<Genere>top = new ArrayList<Genere>();
  for(int i = 0;i < numGeneri; i++)
  {
   if(generi[i].getNumero() >= soglia)
     top.add(generi[i]);
  }
  return top;
 }
}
```

```
public class LettoreVideoTester
{
 public static void main(String[] args)
 {
  LettoreVideo sony = new LettoreVideo();
  sony.aggGenere("horror");
  sony.aggGenere("azione");
  sony.aggGenere("fantasy");
  sony.aggVideo("thor","fantasy");
  sony.aggVideo("come d'incanto","romantico");
```

```
  sony.aggVideo("harry potter","fantasy");
  sony.aggVideo("il signore degli anelli","fantasy");
  sony.aggVideo("matrix","azione");
  sony.aggVideo("skyfall","azione");
  sony.aggVideo("saw","horror");
  sony.playVideo("matrix");
  sony.playVideo("skyfall");
  sony.playVideo("saw");
  System.out.println(sony.contaVideo("fantasy"));
  System.out.println(sony.contaVideo("azione"));
  System.out.println("prova top:");
  System.out.println(sony.top(1));
  sony.rimuoviGenere("fantasy");
  System.out.println(sony.top(2));
 }
}
```

Corriere-19/01/13

Testo:

Un corriere espresso desidera gestire in modo automatico le proprie spedizioni. Si assuma la classe Destinazione, avente come variabile d'istanza citta(stringa), e la classe Pacco, avente come variabili d'istanza codice (stringa), peso(intero) e destinazione riferimento a (Destinazione}. Si assumano gli usuali metodi costruttori, accessori e modificatori Sviluppare la seguente classe:

```
  public class Corriere {
    private ArrayList<Destinazione> destinazioni;
    private ArrayList<Pacco> pacchi;
    public Corriere(){...}
    public void aggiungiDestinazione(String unaCitta){...}
    public void aggiungiPacco(String unCodice, int unPeso, String
unaDestinazione){...}
    public void rimuoviPacco(String unCodice){...}
    public void scambiaDestinazioni(String primoCodice, String
secondoCodice){...}
    public void pesoDestinazione(String unaCitta){...}
    public void spedisciDestinazione(String unaCitta){...}
    public int pesoMedio(){...}
  }
}
```

I metodi di inserimento aggiungono oggetti solamente se questi non sono già presenti. Inoltre, il metodo aggiungiPacco crea un nuovo oggetto di classe Pacco avente un riferimento all'oggetto di classe Destinazione associato alla città specificata come parametro esplicito; tale oggetto deve essere creato se non è già presente nel sistema. Il metodo rimuoviPacco rimuove l'oggetto di classe Pacco avente il codice specificato. Il metodo scambiaDestinazioni scambia le destinazioni dei pacchi aventi codice specificato. Il metodo pesoDestinazione calcola il peso totale dei pacchi da spedire alla città specificata. Il metodo spedisciDestinazione simula una spedizione alla città specificata, rimuovendo dal sistema tutti i pacchi assegnati a tale destinazione, e rimuovendo anche la destinazione stessa.
Infine, il metodo pesoMedio restituisce il peso medio delle spedizioni presenti nel sistema, calcolato come la media sui pesi di ciascuna destinazione.

Consiglio:

In questo esercizio la difficoltà è gestire un ArrayList, cosa che viene spesso richiesta negli esercizi d'esame. Per questo il presente esercizio verrà ampliamente commentato al fine di renderlo il più chiaro possibile.

```
La classe Destinazione non presenta alcuna difficoltà
public class Destinazione
{
private String citta;
/**Crea un oggetto della classe Destinazione.*/
 public Destinazione(String unaCitta)
 {
  citta=unaCitta;
 }
/**Restituisce il nome della destinazione.
 @return Nome della destinazione.*/
 public String getCitta()
 {
  return citta;
 }
}
```

```
La classe Pacco non presenta alcuna difficoltà
public class Pacco
{
private String codice;
private int peso;
private Destinazione destinazione;
/**Crea un oggetto di tipo Pacco.
@unCodice Codice.
@unPeso Peso.
@param unaDestinazione Destinazione.*/
 public Pacco(String unCodice, int unPeso, Destinazione unaDestinazione)
 {
  codice=unCodice;
  peso=unPeso;
  destinazione=unaDestinazione;
 }
/**Restituisce un oggetto della classe destinazione.
 @return Oggetto di classe destinazione.*/
 public Destinazione getDestinazione()
 {
  return destinazione;
 }
/**Cambia la destinazione a un oggetto di classe Pacco.
 @param unaDestinazione La nuova destanazione che verrà assegnata al pacco.*/
 public void setDestinazione(Destinazione unaDestinazione)
 {
  destinazione=unaDestinazione;
 }
/**Restituisce il codice del pacco.
 @return Il codice del pacco.*/
 public String getCodice()
 {
  return codice;
 }
/**Restituisce il peso del pacco.
 @return Il peso del pacco.*/
 public int getPeso()
 {
  return peso;
 }
}
```

La classe Corriere presenta varie difficoltà

```
import java.util.ArrayList;
public class Corriere
{
private ArrayList<Destinazione> destinazioni;
private ArrayList<Pacco> pacchi;
```

Il costruttore deve inizializzare le variabili di istanza.

```
/**Crea un oggetto della classe Corriere.*/
 public Corriere ()
 {
  destinazioni = new ArrayList <Destinazione>();
  pacchi = new ArrayList <Pacco>();
 }
```

Vengono creati due metodi ausiliari per semplificare la stesura dei metodi richiesti, il primo metodo ausiliario cerca una destinazione nell'ArrayList destinazioni, restituisce l'indece in corrispondenza del quale si trova l'oggetto cercato(se presente), nel caso in cui l'oggetto non venga trovato restituisce -1.

```
/**Metodi ausiliari.
 Cerca una destinazione nell' ArrayList destinazioni.
 @param unaCittà Nome della destinazione da cercare.
 @return Indice corrispondente alla destinazione trovata.*/
 private int cercaDestinazione(String unaCitta)
 {
  int result =-1;
  for (int i=0; i<destinazioni.size(); i++)
  {
   if (destinazioni.get(i).getCitta().equalsIgnoreCase(unaCitta))
     result = i ;
  }
  return result;
 }
```

Il secondo metodo ausiliario cerca il pacco dall'ArrayList pacchi, restituisce l'indece in corrispondenza del quale si trova l'oggetto cercato(se presente), nel caso in cui l'oggetto non venga trovato restituisce -1.

```
/**Cerca un pacco.
 @param unaCodice Codice del pacco da cercare.
 @return Indice corrispondente al pacco trovato.*/
 private int cercaPacco(String unCodice)
 {
  int result = -1;
  for (int i=0; i<pacchi.size();i++)
  {
   if(pacchi.get(i).getCodice().equalsIgnoreCase(unCodice))
     result =i;
  }
  return result;
 }
```

La prima istruzione controlla che la destinazione non sia già presente nell'ArrayList dastinazioni(per controllare si usa il metodo ausiliario cercaDestinazione). Se la destinazione è già presente esce dal metodo(return), altrimenti crea un oggetto della classe Destinazione e lo inserisce nell'ArrayList destinazioni.

```
/**Aggiunge un oggetto della classe Destinazione all'ArrayList destinazioni.
 @param unaCitta Nome della destanzazione da aggiungere.*/
 public void aggiungiDestinazione(String unaCitta)
 {
  if (cercaDestinazione(unaCitta)!=-1)
    return;
  Destinazione temp = new Destinazione(unaCitta);
  destinazioni.add(temp);
 }
```

Le prime istruzioni verificano che il pacco e la destinazione non siano già presenti. Se il pacco è già presente nell'ArrayList pacchi esce dal metodo(return). Se il pacco non è presente ma la destinazione è già presente crea il pacco e lo inserisce nell'ArrayList pacchi. Se non sono presenti ne la destinazione ne il pacco crea entrambi e li inserisce nelle rispettive liste: la destinazione viene aggiunta richiamando il metodo aggiungiDestinazione spiegato sopra, il pacco viene creato ed aggiunto all'ArrayList pacchi.

```
/**Aggiunge un oggetto della classe Pacco all'ArrayList pacchi.
 @param unaCodice Codice del pacco da aggiungere.
 @param unPeso Il peso del pacco.
 @param unaDestinazione Nome della destinazione.*/
 public void aggiungiPacco (String unCodice, int unPeso,String unaDestinazione)
 {
  if(cercaPacco(unCodice)!= -1)
    return;
  if(cercaDestinazione(unaDestinazione)!=-1)
  {
   Pacco b = new Pacco(unCodice, unPeso,
destinazioni.get(cercaDestinazione(unaDestinazione)));
   pacchi.add(b);
  }
  if(cercaDestinazione(unaDestinazione)==-1)
  {
   aggiungiDestinazione(unaDestinazione);
   Pacco a = new Pacco (unCodice, unPeso,
destinazioni.get(cercaDestinazione(unaDestinazione)));
   pacchi.add(a);
  }
 }
```

La prima istruzione controlla che il pacco sia presente nell'ArrayList pacchi, se non è presente esce dal metodo(return), altrimenti rimuove il pacco dall'Arraylist pacchi.

```
/**Rimuove il pacco con codice specificato.
 @param unCodice Codice del pacco da rimuovere.*/
 public void rimuoviPacco (String unCodice)
 {
  if(cercaPacco(unCodice)== -1)
    return;
  pacchi.remove(cercaPacco(unCodice));
 }
```

La prima istruzione controlla che i pacchi siano presenti nell'ArrayList pacchi, se non sono presenti esce dal metodo(return). Vengono create due variabili di tipo Destinazione, first contiene la destinazione del primo pacco(che viene estratta dall'ArrayList, sfruttando il primoCodice passato come parametro esplicito), second contiene la destinazione del secondo pacco(che viene estratta dall'ArrayList, sfruttando il primoCodice passato come parametro esplicito). Le ultime istruzioni modificano la destinazione riferita al pacco usando il metodo setDestinazione della classe Pacco.

```
/**Scambia le destinazioni dei due pacchi aventi il codice specificato.
 @param primoCodice Codice del primo pacco.
 @param secondoCodice Codice del secondo pacco.*/
 public void scambiaDestinazione(String primoCodice, String secondoCodice)
 {
  if(cercaPacco(primoCodice)==-1||cercaPacco(secondoCodice)==-1)
    return;
  Destinazione first= pacchi.get(cercaPacco(primoCodice)).getDestinazione();
  Destinazione second= pacchi.get(cercaPacco(secondoCodice)).getDestinazione();
  pacchi.get(cercaPacco(primoCodice)).setDestinazione(second);
  pacchi.get(cercaPacco(secondoCodice)).setDestinazione(first);
 }
```

La prima istruzione controlla che la destinazione sia presente nell'ArrayList destinazioni, se non è presente esce dal metodo(return). Viene inizializzata una variabile int sum necessaria per il calcolo. Il ciclo for visita tutti gli oggetti presenti nell'ArrayList pacchi ed estrae da essi il nome della destinazione, che viene confrontato con la stringa passata come parametro esplicito. Se le stringhe sono uguali la variabile num viene incrementata di una quantità pari al peso del pacco.

```
/**Restituisce il peso dei pacchi da spedire alla destinazione specificata.
 @param unaCitta Destinazione specificata.
 @return Peso di tutti i pacchi da spedire alla destinazione specificata.*/
 public int pesoDestinazione(String unaCitta)
 {
  if(cercaDestinazione(unaCitta)==-1)
    return-1;
  int sum=0;
  for(int i=0;i<pacchi.size();i++)
  {
   if(pacchi.get(i).getDestinazione().getCitta().equalsIgnoreCase(unaCitta))
     sum=sum+pacchi.get(i).getPeso();
  }
  return sum;
 }
```

La prima istruzione controlla che la destinazione sia presente nell'ArrayList destinazioni, se non è presente esce dal metrodo (return). Il ciclo for visita(dall'ultimo al primo, poiché potrebbero esserci più elementi contigui da cancellare, si invita a rivedere l'esercizio 7.1)tutti gli oggetti presenti nell'ArrayList pacchi ed estrae da essi il nome della destinazione che viene confrontato con la stringa passata come parametro esplicito. Se le stringhe sono uguali il pacco viene rimosso dall'ArrayList pacchi. L'ultima istruzione elimina la destinazione dall'ArrayList destinazioni.

```
/**Rimuove tutti i pacchi assegnati a una certa destinazione, e rimuovendo anche
la destinazione stessa.
 @param unaCitta Destinazione dalla quale rimuovere i pacchi.*/
 public void spedisciDestinazione(String unaCitta)
 {
  if(cercaDestinazione(unaCitta)==-1)
    return;
  for(int i=pacchi.size()-1;i>-1;i--)
  {
   if(pacchi.get(i).getDestinazione().getCitta().equalsIgnoreCase(unaCitta))
     pacchi.remove(i);
  }
  destinazioni.remove(cercaDestinazione(unaCitta));}
```

Le prime istruzioni inizializzano due variabili double somma e peso, uguali a zero, necessarie per il calcolo che viene effettuato richiamando il metodo pesoDestinazione. Il ciclo for visita tutti gli oggetti presenti nell'Arraylist destinazioni ed estrae da essi il nome della destinazione, che viene utilizzata come parametro esplicito nel metodo pesoDestinazione. Le ultime istruzioni dividono la somma totale per il numero di destinazioni, per avere il peso medio.

```
/**Restituisce il peso medio delle spedizioni presenti nel sistema.
 @return Peso medio.*/
 public double pesoMedio()
 {
  double somma=0;
  double peso=0;
  for(int i=0; i<destinazioni.size();i++)
  {
   somma=somma+pesoDestinazione(destinazioni.get(i).getCitta());
  }
  peso=somma/destinazioni.size();
  return peso;
 }
}
```

```
import java.util.ArrayList;
public class CorriereTester
{
 public static void main(String [] args)
 {
  Corriere truck = new Corriere();
  truck.aggiungiDestinazione("Padova");
  truck.aggiungiDestinazione("Vicenza");
  truck.aggiungiDestinazione("Milano");
  truck.aggiungiDestinazione("Vicenza");
  truck.aggiungiPacco("0001", 10, "Padova");
  truck.aggiungiPacco("0002", 12, "Vicenza");
  truck.aggiungiPacco("0003", 18, "Vicenza");
  truck.aggiungiPacco("0004", 67, "Milano");
  System.out.println(truck.pesoDestinazione("Vicenza"));
  truck.scambiaDestinazione("0001","0003");
  System.out.println(truck.pesoDestinazione("Vicenza"));
  truck.spedisciDestinazione("Padova");
  truck.rimuoviPacco("0002");
  System.out.println(truck.pesoMedio());
 }
}
```

DittaRiparazioni-20/01/17

Testo:

Una ditta per la riparazione di caldaie desidera gestire elettronicamente i propri interventi a domicilio. A tal fine si consideri la classe Riparazione, avente come variabili d'istanza
indirizzo (stringa) e priorita (int), dove la seconda variabile `e un numero non negativo che rappresenta la priorit`a della riparazione. Si consideri inoltre la classe Tecnico, avente come variabili d'istanza
nome (stringa) e riparazione (riferimento ad un oggetto di classe Riparazione). Si assuma siano gi`a
disponibili gli usuali metodi costruttori, accessori e modificatori per le due classi sopra indicate, senza
sviluppare il codice per queste classi. Sviluppare tutti i metodi della classe sotto riportata.

```
public class DittaRiparazioni {
private ArrayList<Riparazione> riparazioni;
```

```
private Tecnico[] tecnici;
private int numeroTecnici;
public DittaRiparazioni(int maxTecnici){...}
public boolean aggRiparazione(String unIndirizzo, int unaPriorita){...}
public ArrayList<Riparazione> riparazioniInAttesa(){...}
public Riparazione prossimaRiparazione(){...}
public void assegnaRiparazione(){...}
public void terminaRiparazione(String unNome){...}
public boolean aggTecnico(String unNome){...}
public void ferie(ArrayList<String> listaNomi){...}
}
```

La variabile d'istanza riparazioni rappresenta la lista di richieste di riparazioni nel sistema. La variabile d'istanza tecnici rappresenta la lista di tecnici della ditta e deve essere gestita come un array parzialmente riempito, con variabile "soglia" numeroTecnici. La dimensione massima dell'array viene impostata dal metodo costruttore. Nel sistema non devono mai essere presenti pi`u riparazioni con lo stesso indirizzo oppure pi`u tecnici con lo stesso nome.
Il metodo aggRiparazione aggiunge al sistema una nuova richiesta di riparazione. Il metodo restituisce true solo se l'operazione `e eseguita con successo. Il metodo riparazioniInAttesa restituisce la lista delle riparazioni nel sistema non ancora assegnate ad un tecnico. Il metodo prossimaRiparazione restituisce la riparazione con pi`u alta priorit`a non ancora assegnata ad un tecnico (risolvere arbitrariamente i casi di parit`a). Il metodo assegnaRiparazione assegna ad un qualsiasi tecnico libero, se ne esiste almeno uno, la riparazione con pi`u alta priorit`a tra quelle non ancora prese in carico. Non assegnare mai pi`u di un tecnico ad una singola riparazione. Il metodo terminaRiparazione libera il tecnico avente il nome specificato dalla propria riparazione , e rimuove dal sistema la riparazione stessa. Il metodo aggTecnico aggiunge un nuovo tecnico al sistema solo se il numero di riparazioni in attesa supera la met`a del numero totale di riparazioni nel sistema. Il metodo restituisce true solo se l'operazione `e eseguita con successo. Infine, il metodo ferie rimuove dal sistema tutti i tecnici il cui nome appare nella lista passata come argomento esplicito, e rimuove dal sistema anche le riparazioni ad essi assegnate.

Consiglio:

Il metodo più difficile è riparazioniInAttesa, per avere una lista di prenotazioni che non sono assegnate ad alcun tecnico è necessario creare un nuovo ArrayList libere e copiare all'interno di esso tutto il contenuto dell'Arraylist riparazioni(si usi il metodo adAll(ArrayList da_inserire)). Successivamente si tolgonono dall'ArrayList libere tutte le riparazioni che risultano già assegnate a un tecnico.

```
public class Riparazione
{
private String indirizzo;
private int priorita;
    /**Crea un oggetto della classe Riparazione.
     @param unIndirizzo indirizzo.
     @param unaPriorità priorità dell'intervento.*/
    public Riparazione(String unIndirizzo, int unaPriorita)
    {
      indirizzo=unIndirizzo;
      priorita=unaPriorita;
    }
```

```
    /**Restituisce l'indirizzo della riparazione.
     @return indirizzo.*/
    public String getIndirizzo()
    {
      return indirizzo;
    }
    /**Restituisce la priorità dell'intervento.
     @return priorità intervento.*/
    public int getPriorita()
    {
      return priorita;
    }
}


public class Tecnico
 {
private String nome;
private Riparazione riparazione;
    /**Costruisce un oggetto della classe Tecnico.
     @param unNome nome tecnico.
     @param unaRiparazione oggetto di classe Riparazione.*/
    public Tecnico(String unNome, Riparazione unaRiparazione)
    {
      nome=unNome;
      riparazione=unaRiparazione;
    }
    /**Restituisce il nome del tecnico.
     @return nome tecnico.*/
    public String getNome()
    {
      return nome;
    }
    /**Restituisce la riparazione assegnata al tecnico.
     @return oggetto riparazione.*/
    public Riparazione getRiparazione()
    {
      return riparazione;
    }
    /**Cambia la riparazione assegnata al tecnico.
     @param nuovaRiparazione riparazione nuova.*/
    public void setRiparazione(Riparazione nuovaRiparazione)
    {
      riparazione=nuovaRiparazione;
    }
}



import java.util.ArrayList;
public class DittaRiparazioni
 {
private ArrayList<Riparazione> riparazioni;
private Tecnico[] tecnici;
private int numeroTecnici;
    /**Costruisce un oggetto della classe DittaRiparazioni.
     @param maxTecnici dimensione array tecnici.*/
    public DittaRiparazioni(int maxTecnici)
    {
      tecnici=new Tecnico[maxTecnici];
      riparazioni= new ArrayList<Riparazione>();
      numeroTecnici=0;
    }
```

```
 /**Metodo ausiliario che cerca un tecnico nell'array tecnici.
  @param unNome nome tecnico.
  @return indice.*/
 private int cercaT(String unNome)
 {
 int result=-1;
   for(int i=0;i<numeroTecnici;i++)
   {
         if(tecnici[i].getNome().equals(unNome))
               result=i;
   }
 return result;
 }
/**Metodo ausiliario che cerca una riparazione nell'array riparazioni.
@param unIndirizzo indirizzo riparazione.
@return indice.*/
private int cercaR(String unIndirizzo)
{
  int result=-1;
    for(int i=0;i<riparazioni.size();i++)
    {
     if(riparazioni.get(i).getIndirizzo().equals(unIndirizzo))
       result=i;
    }
  return result;
}
 /**Aggiunge una riparazione all'array riparazioni.
  @param unIndirizzo indirizzo riparazione.
  @param unaPriorita priorità della riparazione.
  @return true o false.*/
 public boolean aggRiparazione(String unIndirizzo, int unaPriorita)
 {
   if(cercaR(unIndirizzo)!=-1)
         return false;

   Riparazione a=new Riparazione(unIndirizzo,unaPriorita);
         riparazioni.add(a);
         return true;

 }
 /**Restituisce la lista delle riparazioni in attesa.
  @return lista riparazioni in attesa.*/
 public ArrayList<Riparazione> riparazioniInAttesa()
 {
   ArrayList<Riparazione> a=new ArrayList<Riparazione>();
   a.addAll(riparazioni);
   for(int i=0;i<numeroTecnici;i++)
   {
     for(int j=0;j<a.size();j++)
     {
     if(tecnici[i].getRiparazione()==a.get(j))
         a.remove(j);
     }
   }
 return a;
 }
 /**Restituisce la prossima riparazione.
  @return oggetto Riparazione.*/
```

```
public Riparazione prossimaRiparazione()
 {
   Riparazione a=null;
   int max=0;
   for(int i=0;i<riparazioni.size();i++)
   {
     if(riparazioni.get(i).getPriorita()>max)
     {
      max=riparazioni.get(i).getPriorita();
      a=riparazioni.get(i);
     }
   }
   return a;
 }
/**Assegna una riparazione a un tecnico libero.*/
 public void assegnaRiparazione()
 {
  ArrayList<Riparazione> libere= riparazioniInAttesa();
  if(libere.size()==0)
   return;
   for(int i=0;i<numeroTecnici;i++)
   {
    if(tecnici[i].getRiparazione()==null)
        {
         tecnici[i].setRiparazione(prossimaRiparazione());
         return;
        }
    }
   }
 /**libera il tecnico avente il nome specificato dalla propria riparazione
  e rimuove dal sistema la riparazione stessa.
  @param unNome nome tecnico.*/
 public void terminaRiparazione(String unNome)
 {
   String terminata="";
   if(cercaT(unNome)==-1)
         return;
         for(int i=0;i<numeroTecnici;i++)
         {
          if(tecnici[i].getNome().equals(unNome))
           {
               terminata=tecnici[i].getRiparazione().getIndirizzo();
               tecnici[i].setRiparazione(null);
           }
         riparazioni.remove(cercaR(terminata));
        }
 }
 /**aggiunge un nuovo tecnico al sistema solo se il numero di riparazioni in
 attesa supera la met`a del numero totale di riparazioni nel sistema.
 @param unNome nome tecnico.
 @return true o false.*/
 public boolean aggTecnico(String unNome)
 {
  if(riparazioni.size()/2>riparazioniInAttesa().size())
   return false;
  if(tecnici.length<=numeroTecnici)
   return false;

   tecnici[numeroTecnici]=new Tecnico(unNome,null);
   numeroTecnici++;
   return true;
 }
```

```
    /**Rimuove dal sistema tutti i tecnici il cui nome appare nella lista
passata come argomento esplicito e rimuove dal sistema anche le
riparazioni ad essi assegnate.
    @param listaNomi lista dei nomi da rimuovere.*/
     public void ferie(ArrayList<String> listaNomi)
     {
       if(listaNomi.size()==0)
            return;
       for(int i=0;i<listaNomi.size();i++)
       {
         terminaRiparazione(listaNomi.get(i));
       }

       for(int i=0;i<listaNomi.size();i++)
       {
         for(int j=0;j<numeroTecnici;j++)
         {
           if(listaNomi.get(i).equals(tecnici[j].getNome()))
           {
             tecnici[j]=tecnici[numeroTecnici-1];
             numeroTecnici--;
           }
         }
       }
      }
}



public class DittaRiparazioniTester
{
 public static void main(String[] args)
  {
  DittaRiparazioni pizzatosnc=new DittaRiparazioni(3);
  System.out.println("inserisco riparazioni:");
  System.out.println(pizzatosnc.aggRiparazione("contrà barche",3));
  System.out.println(pizzatosnc.aggRiparazione("stradella san nicola",1));
  System.out.println(pizzatosnc.aggRiparazione("viale margherita",2));
  System.out.println("guardo le riparazioni libere:");
  System.out.println(pizzatosnc.riparazioniInAttesa());
  System.out.println("vedo la prossima riparazione da fare:");
  System.out.println(pizzatosnc.prossimaRiparazione());
  System.out.println("aggiungo un tecnico:");
  System.out.println(pizzatosnc.aggTecnico("Bressan Marco"));
  System.out.println("vedo quante riparazioni sono rimaste libere:");
  System.out.println(pizzatosnc.riparazioniInAttesa());
  System.out.println("assegno una riparazione al tecnico.");
  pizzatosnc.assegnaRiparazione();
  System.out.println("vedo quante riparazioni sono rimaste libere:");
  System.out.println(pizzatosnc.riparazioniInAttesa());
  System.out.println("la riparazione è terminata, ne assegno un altra.");
  pizzatosnc.terminaRiparazione("Bressan Marco");
  pizzatosnc.assegnaRiparazione();
  System.out.println("vedo quante riparazioni sono rimaste libere:");
  System.out.println(pizzatosnc.riparazioniInAttesa());
  }
}
```

CarSharing-19/09/16

Testo:

Una societa' di *car sharing* intende dotarsi di un programma per gestire il proprio parco automobili. Si assumano le classi Parcheggio, avente la variabile d'istanza nome (stringa), e Automobile,
avente le variabili d'istanza targa (stringa), viaggi (intero), e locazione (riferimento a Parcheggio).
Assumere gli usuali metodi di accesso e modifica per tali classi senza sviluppare il codice.
Sviluppare tutti i metodi della classe CarSharing, definita come di seguito:

```
public class CarSharing {
private ArrayList<Parcheggio> parcheggi;
private ArrayList<Automobile> automobili;
public CarSharing(){...};
public void aggiungiParcheggio(String unNome){...};
public void aggiungiAutomobile(String unaTarga, String unNomeParcheggio){...};
public void transito(String unaTarga, String parcheggioArrivo){...};
public int contaAutomobili(String unNomeParcheggio){...};
public void rimuoviAutomobili(int unNumeroViaggi){...};
public String statistica(){...};
};
```

Il metodo aggiungiParcheggio aggiunge un nuovo parcheggio, senza creare duplicati. Il metodo
aggiungiAutomobile aggiunge una nuova automobile senza creare duplicati, e la assegna al parcheggio
indicato. Creare il parcheggio se questo non esiste gi`a; la nuova automobile deve avere un numero
di viaggi pari a zero. Il metodo transito sposta l'automobile indicata ad un nuovo parcheggio, ed
incrementa di una unit`a il numero di viaggi. Il metodo contaAutomobili restituisce il numero di
automobili presenti nel parcheggio indicato. Il metodo rimuoviAutomobili rimuove dal sistema tutte
le automobile che abbiano compiuto un numero di viaggi uguale o superiore alla quantit`a indicata.
Il metodo statistica restituisce il nome del parcheggio avente il massimo numero di automobili; risolvere arbitrariamente i casi di parita'.

Consiglio:

Il metodo più difficile è statistica, è necessario utilizzare un for innestato dove il più esterno scorre la lista parcheggi e il più interno scorre la lista automobili.

```
public class Parcheggio
{
  private String nome;
    /**Crea un oggetto della classe Parcheggio.
     @param unNome nome del parcheggio.*/
    public Parcheggio(String unNome)
    {
      nome=unNome;
    }
    /**Restituisce il nome del parcheggio.
     @return nome del parcheggio.*/
    public String getNome()
    {
      return nome;
    }
}
```

```
public class Automobile
{
      private String targa;
      private int viaggi;
      private Parcheggio locazione;
    /**Crea un oggetto della classe Automobile.
     @param unaTarga targa veicolo.
     @param unViaggio viaggi effettuati.
     @param unParck oggetto di tipo Parcheggio.*/
    public Automobile(String unaTarga, int unViaggio, Parcheggio unParck)
    {
     targa=unaTarga;
     viaggi=unViaggio;
     locazione=unParck;
    }
    /**Restituisce la targa del veicolo.
     @return targa.*/
    public String getTarga()
    {
      return targa;
    }
    /**Restituisce i viaggi del veicolo.
     @return viaggi.*/
    public int getViaggi()
    {
      return viaggi;
    }
    /**Restituisce il parcheggio in cui si trova il veicolo.
     @return oggetto Parcheggio.*/
    public Parcheggio getLocazione()
    {
      return locazione;
    }
    /**Modifica il parcheggio associato all'automobile.
     @param nuovo nuovo parcheggio.*/
    public void setParcheggio(Parcheggio nuovo)
    {
      locazione=nuovo;
      viaggi++;
    }
}



import java.util.ArrayList;
public class CarSharing
{
private ArrayList<Parcheggio> parcheggi;
private ArrayList<Automobile> automobili;
    /**Costruisce un oggetto della classe CarSharing.*/
    public CarSharing()
    {
     parcheggi=new ArrayList<Parcheggio>();
     automobili=new ArrayList<Automobile>();
    }
    /**Metodo ausiliario che cerca un auto nella lista automobili.
     @param unaTarga targa veicolo da cercare.
     @return indice.*/
```

```
    public int cercaAuto(String unaTarga)
    {
     int result=-1;
     for(int i=0;i<automobili.size();i++)
     {
      if(automobili.get(i).getTarga().equals(unaTarga))
            result=i;
     }
     return result;
    }
     /**Metodo ausiliario che cerca un parcheggio nella lista parcheggi.
     @param unNome nome parcheggio.
     @return indice.*/
    public int cercaPark(String unNome)
    {
     int result=-1;
     for(int i=0;i<parcheggi.size();i++)
     {
      if(parcheggi.get(i).getNome().equals(unNome))
            result=i;
     }
     return result;
    }
    /**Aggiunge un nuovo parcheggio.
     @param unNome nome parcheggio.*/
    public void aggiungiParcheggio(String unNome)
    {
      if(cercaPark(unNome)!=-1)
            return;
      parcheggi.add(new Parcheggio(unNome));
    }
    /**Aggiunge una nuova automobile senza creare duplicati, e la assegna al
parcheggio
     indicato.
     @param unaTarga targa veicolo.
     @param unNomeParcheggio nome parcheggio.*/
    public void aggiungiAutomobile(String unaTarga,String unNomeParcheggio)
    {
      if(cercaAuto(unaTarga)!=-1)
            return;
      if(cercaPark(unNomeParcheggio)!=-1)
            automobili.add(new Automobile(unaTarga, 0,
parcheggi.get(cercaPark(unNomeParcheggio))));
      else
      {
      Parcheggio a= new Parcheggio(unNomeParcheggio);
       parcheggi.add(a);
       automobili.add(new Automobile(unaTarga,0,a));

      }
    }
    /**Sposta l'automobile indicata ad un nuovo parcheggio.
     @param unaTarga targa veicolo.
     @param parcheggioArrivo parcheggio di destinazione.*/
    public void transito(String unaTarga, String parcheggioArrivo)
    {
     if(cercaAuto(unaTarga)==-1||cercaPark(parcheggioArrivo)==-1)
            return;

automobili.get(cercaAuto(unaTarga)).setParcheggio(parcheggi.get(cercaPark(parche
ggioArrivo)));
    }
```

```
   /**Restituisce il numero di automobili presenti nel parcheggio indicato.
     @param unNomeParcheggio nome parcheggio.
     @return numero auto.*/
    public int contaAutomobili(String unNomeParcheggio)
    {
     if(cercaPark(unNomeParcheggio)==-1)
      return -1;
     int tot=0;
      for(int i=0;i<automobili.size();i++)
      {
        if(automobili.get(i).getLocazione().getNome().equals(unNomeParcheggio))
          tot++;
      }
     return tot;
    }
    /**Rimuove dal sistema tutte le automobile che abbiano compiuto un numero di
viaggi uguale o superiore alla quantit`a indicata.
     @param unNumeroViaggi numero soglia.*/
    public void rimuoviAutomobili(int unNumeroViaggi)
    {
      for(int i=automobili.size()-1;i>=0;i--)
      {
       if(automobili.get(i).getViaggi()>=unNumeroViaggi)
            automobili.remove(i);
      }
    }
/**Restituisce il nome del parcheggio avente il massimo numero di
automobili.
@return parcheggio più affollato.*/
    public String statistica()
    {
     int max=0;
     int indice=-1;
      for(int i=0;i<parcheggi.size();i++)
      {
            int cont=0;
            for(int j=0;j<automobili.size();j++)
            {

if(automobili.get(j).getLocazione().getNome().equals(parcheggi.get(i).getNome())
             cont++;
             if(cont>max)
             {
              max=cont;
              indice=i;
             }
            }
      }
      return parcheggi.get(indice).getNome();
    }
}
```

Magazzino-02/03/13

Testo:

Un negozio di strumenti musicali desidera gestire in modo automatico il proprio magazzino.
Si assuma la classe Articolo, avente come variabili d'istanza codice (stringa) e quantitativo(intero). Si assumano gli usuali metodi costruttori, accessori e modificatori per la classe Articolo.
Sviluppare la seguente classe:

```
  public class Magazzino {
```

```
    private ArrayList<Articolo> disponibili;
    private ArrayList<Articolo> ordini;

    public Magazzino(){...}
    public void ordinaArticolo(String unCodice, int unQuantitativo){...}
    public void arrivoArticolo(String unCodice){...}
    public void vendiArticolo(String unCodice, int unQuantitativo){...}
    public int minimoQuantitativo(){...}
    public void rifornisci(int n){...}
    public int contaArticoli(String unPrefisso){...}
  }
}
```

I due ArrayList disponibili e ordini contengono rispettivamente gli articoli disponibili nel magazzino e gli articoli ordinati al produttore ma non ancora disponibili nel magazzino.
Il metodo ordinaArticolo aggiunge un oggetto articolo agli ordini, se non esiste già, oppure somma il quantitativo specificato al quantitativo dell'articolo già esistente nella lista degli ordini.
Il metodo arrivoArticolo rimuove dalla lista degli ordini l'oggetto articolo avente il codice specificato, e somma il quantitativo dell'ordine al quantitativo dell'articolo presente nei disponibili; se l'articolo non è presente nei disponibili, crearne uno nuovo con il quantitativo dell'ordine.
Il metodo vendiArticolo sottrae il quantitativo specificato da un oggetto articolo presente nei disponibili, e rimuove l'articolo dai disponibili se tutto il quantitativo viene esaurito. Inoltre, se la richiesta di vendita è superiore alla disponibilità, aggiornare la lista degli ordini per questo articolo con un quantitativo pari alla differenza.
Il metodo minimoQuantitativo restituisce il quantitativo più basso tra tutti i quantitativi associati agli articoli presenti nei disponibili.
Il metodo rifornisci crea un ordine per un quantitativo pari ad n per ciascun articolo presente nei disponibili con un quantitativo minimo, come calcolato da minimoQuantitativo.
Il metodo contaArticoli restituisce il numero totale di articoli presenti nella lista degli ordini o nei disponibili, aventi il codice che inizi con la stringa unPrefisso.

Consiglio:
Si invita ad implementare due metodi ausiliari che verifichino la presenza di un'articolo negli ArrayList disponibili e ordini. Nella classe Articolo si consiglia di implementare dei metodi aumentaQuantità e diminuisciQuantità che aiutano la stesura della classe Magazzino.

```
public class Articolo
{
private String codice;
private int quantitativo;
/**Creo un oggetto della classe Articolo.
 @param unCodice Codice dell'articolo.
 @param unQuantitativo Quantità articolo.*/
 public Articolo(String unCodice, int unQuantitativo)
 {
  codice = unCodice;
  quantitativo = unQuantitativo;
 }
/**Restituisce il codice di un articolo.
 @return Il codice di un articolo.*/
 public String getCodice()
 {
  return codice;
 }
/**Restituisce il quantitativo di un articolo.
 @return Il quantitativo di un articolo.*/
```

```
 public int getQuantita()
 {
  return quantitativo;
 }
/**Aumenta la quantità di un articolo.
 @param unaQuantita Quantità da aggiungere all'articolo.*/
 public void aumentaQuantita(int unaQuantita)
 {
  quantitativo += unaQuantita;
 }
/**Dimunuisce la quantità di un articolo.
 @param unaQuantita Quantità da togliere all'articolo.*/
 public void diminuisciQuantita(int unaQuantita)
 {
  quantitativo -= unaQuantita;
 }
}



import java.util.ArrayList;
public class Magazzino
{
private ArrayList<Articolo> disponibili;
private ArrayList<Articolo> ordini;
/**Creo un oggetto della classe Magazzino.*/
 public Magazzino()
 {
  disponibili=new ArrayList<Articolo>();
  ordini=new ArrayList<Articolo>();
 }
/**Metodo ausiliario che restituisce l'indice dell'articolo specificato.
 @param unCodice Codice articolo.
 @return Indice.*/
 private int cercaCodiceO(String unCodice)
 {
  int result=-1;
  for(int i=0;i<ordini.size();i++)
  {
   if(ordini.get(i).getCodice().equals(unCodice))
     result=i;
  }
  return result;
 }
/**Metodo ausiliario che cereca nell' ArrayList disponibili
 l'indice dell' articolo specificato.
 @param unCodice Codice articolo.
 @return Indice.*/
  private int cercaCodiceD(String unCodice)
  {
   int result=-1;
   for(int i=0;i<disponibili.size();i++)
   {
    if(disponibili.get(i).getCodice().equals(unCodice))
      result=i;
   }
   return result;
  }
/**Aggiunge un oggetto articolo agli ordini, se non esiste già,
 oppure somma il quantitativo specificato all'articolo già esistenete.
 @param unCodice Codice articolo.
 @param unaQuantita Quantità articolo.*/
```

```
public void ordinaArticolo(String unCodice, int unaQuantita)
 {
  if(cercaCodiceO(unCodice)!=-1)
    ordini.get(cercaCodiceO(unCodice)).aumentaQuantita(unaQuantita);
  else
  {
   Articolo a=new Articolo(unCodice, unaQuantita);
   ordini.add(a);
  }
 }
/**Rimuove dalla lista ordini l'articolo specificato e somma
 il quantitativo dell'ordine all'articolo presente nei disponibili,
 se l'articolo non è presente viene creato.
 @param unCodice Codice articolo.*/
 public void arrivoArticolo(String unCodice)
 {
  if(cercaCodiceO(unCodice)==-1)
    return;
  int q=ordini.get(cercaCodiceO(unCodice)).getQuantita();
  if(cercaCodiceD(unCodice)!=-1)
    disponibili.get(cercaCodiceD(unCodice)).aumentaQuantita(q);
  else
  {
   Articolo b=new Articolo(unCodice, q);
   disponibili.add(b);
  }
  ordini.remove(cercaCodiceO(unCodice));
 }
/**Sottrae il quantitativo specificato da un articolo.
 @param unCodice Codice articolo.
 @param unaQuantita Quantitativo.*/
 public void vendiArticolo(String unCodice, int unQuantitativo)
 {
  if(cercaCodiceD(unCodice)==-1)
    return;
  int q=disponibili.get(cercaCodiceD(unCodice)).getQuantita();
  if(q>=unQuantitativo)
    disponibili.get(cercaCodiceD(unCodice)).diminuisciQuantita(unQuantitativo);
  else
  {
   int differenza=unQuantitativo-q;
   Articolo c=new Articolo(unCodice,differenza);
   ordini.add(c);
  }
 }
/**Restituisce il quantitativo più basso tra tutti i quantitativi
 associati agli oggetti presenti nella lista disponibili.
 @return Quantitativo più basso.*/
 public int minimoQuantitativo()
 {
  if(disponibili.isEmpty()==true)
    return -1;
  int min=disponibili.get(0).getQuantita();
  for(int i=0;i<disponibili.size();i++)
  {
   int cont=0;
   cont=disponibili.get(i).getQuantita();
   if(cont<min)
     min=cont;
  }
  return min;
 }
```

```
/**Crea un ordine di una quantià pari ad n,
 per ciascun articolo presente nei disponibili con un quantitativo minimo.
 @param n Quantitativo.*/
 public void rifornisci(int n)
 {
  for(int i=0;i<disponibili.size();i++)
  {
  if(disponibili.get(i).getQuantita()==minimoQuantitativo())
    ordinaArticolo(disponibili.get(i).getCodice(),n);
  }
 }
/**Restituisce il numero totale di articoli presenti nella lista
 degli ordini o nei disponibili, aventi il codice che inizia con un prefisso.
 @param unPrefisso Prefisso.
 @return Numero articoli aventi il codice che inizia con un prefisso.*/
 public int contaArticoli(String unPrefisso)
 {
  if(disponibili.isEmpty()==true&&ordini.isEmpty()==true)
    return -1;
  int cont=0;
  for(int i=0;i<disponibili.size();i++)
  {
   if(disponibili.get(i).getCodice().substring(0,1).equals(unPrefisso))
     cont++;
  }
  for(int i=0;i<ordini.size();i++)
  {
   if(ordini.get(i).getCodice().substring(0,1).equals(unPrefisso))
     cont++;
  }
  return cont;
 }
}



public class MagazzinoTester
{
 public static void main(String[] args)
 {
  Magazzino deposito= new Magazzino();
  deposito.ordinaArticolo("0000",4);
  deposito.ordinaArticolo("0001",9);
  deposito.ordinaArticolo("0002",8);
  deposito.ordinaArticolo("0003",34);
  deposito.ordinaArticolo("0004",66);
  System.out.println(deposito.contaArticoli("s"));
  System.out.println(deposito.minimoQuantitativo());
  deposito.arrivoArticolo("0000");
  deposito.arrivoArticolo("0001");
  deposito.arrivoArticolo("0002");
  deposito.arrivoArticolo("0003");
  deposito.arrivoArticolo("0004");
  System.out.println(deposito.minimoQuantitativo());
  System.out.println(deposito.contaArticoli("0"));
  deposito.rifornisci(2);
  deposito.arrivoArticolo("0000");
  System.out.println(deposito.minimoQuantitativo());
 }
}
```

UfficioContabile-18/02/12

Testo:

Simulare mediante un programma Java un ufficio contabile. Si assuma la classe Impiegato, avente come variabile d'istanza nome (stringa), e la classe Pratica, avente come variabili d'istanza codice (stringa), giorniAttesa (intero) rappresentante il numero di giorni trascorsi dalla creazione della pratica, e impiegato, rappresentante un riferimento a Impiegato che ha in carica la pratica. Si assumano gli usuali metodi costruttori, accessori e modificatori per queste due classi senza sviluppare il codice. Sviluppare tutti i metodi della seguente classe:

```
public class UfficioContabile {
private Impiegato[] impiegati;
int numeroImpiegati;
private ArrayList<Pratica> pratiche;
public UfficioContabile(int maxImpiegati){...}
public void nuovoGiorno(){...}
public int cercaImpiegato(String unNomeImpiegato){...}
public int cercaPratica(String unCodice){...}
public void aggiungiPratica(String unCodice, String unNomeImpiegato){...}
public int cercaMaxGiorniAttesa(String unNomeImpiegato){...}
public void svolgiPratica(String unNome){...}
public void aggiungiImpiegato(String unNome){...}
public void eliminaImpiegato(String unNome, String unSostituto){...}
}
```

L'array impiegati deve essere gestito come un array parzialmente riempito, usando come sentinella la variabile numeroImpiegati. Il metodo nuovoGiorno incrementa di una unità la variabile d'istanza giorniAttesa di tutte le pratiche presenti nel sistema. Il metodo aggiungiPratica aggiunge una pratica avente la variabile d'istanza giorniAttesa posta a zero, e con un collegamento all'oggetto Impiegato presente nel sistema avente il nome specificato: aggiungere la pratica solo se l'impiegato associato è presente e non creare duplicati della pratica. Il metodo cercaMaxGiorniAttesa restituisce l'indice della pratica dell'impiegato specificato avente il valore di giorniAttesa più alto. Il metodo svolgiPratica rimuove la pratica assegnata all'impiegato specificato, avente il valore di giorniAttesa più alto. Il metodo aggiungiImpiegato aggiunge un nuovo impiegato, senza creare duplicati e solo se vi è posto nell'array, e gli assegna la pratica presente nel sistema avente il valore di giorniAttesa più alto. Il metodo eliminaImpiegato rimuove l'impiegato indicato, ed assegna tutte le sue pratiche all'impiegato sostituto. Nei metodi sopra riportati, tutti i casi di parità devono essere risolti arbitrariamente.

Consiglio:

Nella classe Pratica si consiglia di implementare dei metodi cambiaImpiegato e aumentaAttesa che aiutano la stesura della classe UfficioContabile. Si prega di fare particolare attenzione al metodo eliminaImpiegato(si rimanda all'esercizio 7.1)

```
public class Impiegato
{
private String nome;
/**Crea un oggetto della classe Impiegato.
 @param unNome Nome impiegato.*/
 public Impiegato(String unNome)
 {
  nome=unNome;
 }
/**Restituisce il nome dell'impiegato.
 @return Il nome dell'impiegato.*/
 public String getNome()
 {
  return nome;
 } }
```

```
public class Pratica
{
private String codice;
private int giorniAttesa;
private Impiegato impiegato;
/**Crea un oggetto della classe Pratica.
 @param unCodice Codice della pratica.
 @param numeroGiorniAttesa Numero giorni passati dalla creazione della pratica.
 @param unImpiegato impiegato che ha in carica la pratica.*/
 public Pratica(String unCodice, int numeroGiorniAttesa, Impiegato unImpiegato)
 {
  codice=unCodice;
  giorniAttesa=numeroGiorniAttesa;
  impiegato=unImpiegato;
 }
 /**Restituisce il codice.
  @return il codice.*/
  public String getCodice()
  {
   return codice;
  }
 /**Restituisce i giorni di attesa.
  @return i giorni di attesa.*/
  public int getGiorniAttesa()
  {
   return giorniAttesa;
  }
 /**Restituisce l'impiegato che ha in carica la pratica.
  @return l'impiegato che ha in carica la pratica.*/
  public Impiegato getImpiegato()
  {
   return impiegato;
  }
 /**Sostituisce l'impiegato che ha in carica la pratica.
  @param nuovoImpiegato il nuovo impiegato a cui verrà data la pratica.*/
  public void cambiaImpiegato(Impiegato nuovoImpiegato)
  {
   impiegato=nuovoImpiegato;
  }


/**Aumenta di una unità i giorni di attesa di una pratica.*/
 public void aumentaAttesa()
 {
  giorniAttesa++;
 }
}


import java.util.ArrayList;
public class UfficioContabile
{
private Impiegato[] impiegati;
private int numeroImpiegati;
private ArrayList<Pratica>pratiche;
/**Crea un oggetto della classe UfficioContabile.
 @param int maxImpiegati*/
 public UfficioContabile(int maxImpiegati)
 {
  impiegati=new Impiegato[maxImpiegati];
  numeroImpiegati=0;
  pratiche=new ArrayList<Pratica>();
 }
```

```
/**Il metodo nuovoGiorno incrementa di una unità la variabile d'istanza
 giorniAttesa di tutte le pratiche presenti nel sistema.*/
 public void nuovoGiorno()
 {
  for(int i=0;i<pratiche.size();i++)
  {
   pratiche.get(i).aumentaAttesa();
  }
 }
/**Restituisce l'indice corrispondente all' impiegato cercato.
 @param unNomeImpiegato Nome dell'impiegato da cercare.
 @return Indice.*/
 public int cercaImpiegato(String unNomeImpiegato)
 {
  int result=-1;
  for(int i=0;i<numeroImpiegati;i++)
  {
   if(impiegati[i].getNome().equals(unNomeImpiegato))
     result=i;
  }
  return result;
 }
/**Restituisce l'indice corrispondente alla pratica cercata.
 @param unCodice Codice della pratica da cercare.
 @return Indice.*/
 public int cercaPratica(String unCodice)
 {
  int result=-1;
  for(int i=0;i<pratiche.size();i++)
  {
   if(pratiche.get(i).getCodice().equals(unCodice))
     result=i;
  }
  return result;
 }
/**Il metodo aggiungiPratica aggiunge una pratica al sistema.
 @param unCodice Codice della pratica.
 @param unNomeImpiegato Impiegato a cui verrà data in carica la pratica.*/
public void aggiungiPratica(String unCodice, String unNomeImpiegato)
 {
  if(cercaImpiegato(unNomeImpiegato)==-1||cercaPratica(unCodice)!=-1)
    return;
  Pratica nuovaPratica=new
Pratica(unCodice,0,impiegati[cercaImpiegato(unNomeImpiegato)]);
  pratiche.add(nuovaPratica);
 }
/**Restituisce l'indice della pratica dell'impiegato specificato avente
 il valore di giorniAttesa più alto.
 @param UnNomeImpiegato Nome dell'impiegato.
 @return indice della pratica avente il valore di giorniAttesa più alto.*/
```

```
public int cercaMaxGiorniAttesa(String unNomeImpiegato)
 {
  if(cercaImpiegato(unNomeImpiegato)==-1)
    return-1;
  int giorniMax=0;
  int indice=-1;
  for(int i=0;i<pratiche.size();i++)
  {
   if(pratiche.get(i).getImpiegato().getNome().equals(unNomeImpiegato))
   {
    if(pratiche.get(i).getGiorniAttesa()>giorniMax)
    {
     giorniMax=pratiche.get(i).getGiorniAttesa();
     indice=i;
    }
   }
  }
  return indice;
 }
/**Rimuove la pratica assegnata all'impiegato specificato,
avente il valore di giorniAttesa più alto.
@param unNome Nome impiegato.*/
 public void svolgiPratica(String unNome)
 {
  if(cercaImpiegato(unNome)==-1||cercaMaxGiorniAttesa(unNome)==-1)
   return;
  pratiche.remove(cercaMaxGiorniAttesa(unNome));
 }
/**Aggiunge un impiegato e gli assegna la pratica i giorniAttesa più alti.
@unNome Nome del nuovo impiegato.*/
 public void aggiungiImpiegato(String unNome)
 {
  if(cercaImpiegato(unNome)!=-1||numeroImpiegati>impiegati.length)
    return;
  Impiegato nuovoImpiegato=new Impiegato(unNome);
  impiegati[numeroImpiegati]=nuovoImpiegato;
  numeroImpiegati++;
  int giorniMax=0;
  int indice=-1;
  for(int i=0;i<pratiche.size();i++)
  {
   if(pratiche.get(i).getGiorniAttesa()>giorniMax)
   {
    giorniMax=pratiche.get(i).getGiorniAttesa();
    indice=i;
   }
  }
  if(pratiche.isEmpty()==true)
    return;
  pratiche.get(indice).cambiaImpiegato(nuovoImpiegato);
 }
/**Rimuove l'impiegato indicato, ed assegna tutte le sue pratiche all'impiegato
sostituto.
@unNome Nome dell'impiegato da sostituire.
@unSostituto Nome dell'impiegato sostituto.*/
```

```
public void eliminaImpiegato(String unNome, String unSostituto)
 {
  if(cercaImpiegato(unNome)==-1||cercaImpiegato(unSostituto)==-1)
    return;
  for(int i=pratiche.size()-1;i>-1;i--)
  {
   if(pratiche.get(i).getImpiegato().getNome().equals(unNome))
     pratiche.get(i).cambiaImpiegato(impiegati[cercaImpiegato(unSostituto)]);
  }
  impiegati[cercaImpiegato(unNome)]=impiegati[numeroImpiegati-1];
  numeroImpiegati--;
 }
}


import java.util.ArrayList;
public class UfficioContabileTester
{
 public static void main(String[] args)
 {
  UfficioContabile ufficio=new UfficioContabile(50);
  ufficio.aggiungiImpiegato("Pizzato");
  ufficio.aggiungiImpiegato("Da Giau");
  ufficio.aggiungiImpiegato("Obradovic");
  ufficio.aggiungiImpiegato("Zecchin");
  System.out.println(ufficio.cercaImpiegato("Pizzato"));
  ufficio.eliminaImpiegato("Zecchin","Obradovic");
  System.out.println(ufficio.cercaImpiegato("Zecchin"));
  ufficio.aggiungiPratica("8435","Pizzato");
  ufficio.nuovoGiorno();
  System.out.println(ufficio.cercaMaxGiorniAttesa("Pizzato"));
  ufficio.aggiungiPratica("0964","Da Giau");
  ufficio.aggiungiPratica("6743","Obradovic");
  ufficio.aggiungiPratica("2350","Da Giau");
  ufficio.nuovoGiorno();
  ufficio.nuovoGiorno();
  System.out.println(ufficio.cercaMaxGiorniAttesa("Pizzato"));
  ufficio.aggiungiImpiegato("Maculan");
  System.out.println(ufficio.cercaMaxGiorniAttesa("Obradovic"));
  ufficio.svolgiPratica("Pizzato");
  System.out.println(ufficio.cercaMaxGiorniAttesa("Da Giau"));
 }
}
```

AssegnazioneTesi-29/06/12

Testo:

Un corso di studi desidera gestire automaticamente l'assegnazione delle tesi di laurea ai propri studenti. Si assuma la classe Tesi, avente come variabile d'istanza titolo (stringa), e la classe Studente, avente come variabili d'istanza matricola (intero) e tesiAssegnata (riferimento a Tesi). Sviluppare tutti i metodi della seguente classe:

```
public class AssegnazioneTesi {
private ArrayList<Studente> studenti;
private ArrayList<Tesi> tesi;
public AssegnazioneTesi(){...}
public void aggiungiTesi(String unTitolo){...}
public void aggiungiStudente(int unaMatricola){...}
public boolean tesiLibera(String unTitolo){...}
public boolean assegnaTesi(int unaMatricola, String unTitolo){...}
public void laureato(int unaMatricola){...}
public void cambioTesi(int unaMatricola, String titoloNuovaTesi){...}
public int statistica(){...}
}
```

Non devono mai essere creati duplicati di tesi o studenti nel sistema. Il metodo aggiungiStudente aggiunge uno studente avente la variabile d'istanza tesiAssegnata posta a null. Il predicato tesiLibera restituisce true se e solo se la tesi indicata non è stata ancora assegnata ad uno studente. Il metodo assegnaTesi associa uno studente alla tesi indicata, solo se quest'ultima _e disponibile; in caso contrario viene restituito il valore false. Il metodo laureato rimuove dal sistema uno studente e la tesi a lui assegnata. Il metodo cambioTesi assegna una nuova tesi ad uno studente e rende libera la vecchia tesi. Il metodo statistica restituisce la percentuale di tesi non ancora assegnate presenti nel sistema, calcolata come rapporto tra il numero di tesi libere ed il numero totale di tesi, e arrotondata ad un intero.

Consigli:
Si invita ad implementare due metodi ausiliari che verifichino la presenza di una tesi e di uno studente nei rispettivi ArrayList. Nella classe Studente si consiglia di implementare un metodo cambiaTesi che aiuta la stesura della classe AssegnazioneTesi.

```
public class Tesi
{
private String titolo;
/**Crea un oggetto della classe Tesi.
 @param unTitolo Titolo della tesi.*/
 public Tesi(String unTitolo)
 {
  titolo=unTitolo;
 }
/**Restituisce il titolo della tesi.
 @return Titolo della tesi.*/
 public String getTitolo()
 {
  return titolo;
 }
}
```

```
public class Studente
{
private int matricola;
private Tesi tesiAssegnata;
/**Crea un oggetto della classe Studente.
 @param unaMatrincola Numero di matricola.*/
 public Studente(int unaMatricola)
 {
  matricola=unaMatricola;
  tesiAssegnata=null;
 }
/**Restituisce il numero di matricola.
 @return Il numero di matricola.*/
 public int getMatricola()
 {
  return matricola;
 }
/**Restituisce la tesi assegnata.
 @return La tesi assegnata.*/
 public Tesi getTesi()
 {
 return tesiAssegnata;
 }
/**Cambia la tesi assegnata.
 @param nuovaTesi Nuova tesi.*/
```

```
 public void sostituisciTesi(Tesi nuovaTesi)
 {
  tesiAssegnata=nuovaTesi;
 }
}



import java.util.ArrayList;
public class AssegnazioneTesi
{
private ArrayList<Studente> studenti;
private ArrayList<Tesi> tesi;
/**Crea un oggetto della classe AssegnazioneTesi.*/
 public AssegnazioneTesi()
 {
  studenti=new ArrayList<Studente>();
  tesi=new ArrayList<Tesi>();
 }
/**Metodi ausiliari.*/
/**Cerca uno studente.
 @param una Matricola Matricola dello studente da cercare.
 @return Indice dello studente.*/
 private int cercaStudente(int unaMatricola)
 {
  int result=-1;
  for(int i=0;i<studenti.size();i++)
  {
  if(studenti.get(i).getMatricola()==unaMatricola)
   result=i;
  }
  return result;
 }
/**Cerca una tesi.
 @param unTitolo Titolo della tesi da cercare.
 @return indice della tesi.*/
 private int cercaTesi(String unTitolo)
 {
  int result=-1;
  for(int i=0;i<tesi.size();i++)
  {
  if(tesi.get(i).getTitolo().equals(unTitolo))
   result=i;
  }
  return result;
 }
/**Aggiunge una tesi al sistema.
 @param unTitolo Titolo della tesi.*/
 public void aggiungiTesi(String unTitolo)
 {
  if(cercaTesi(unTitolo)!=-1)
    return;
  Tesi nuovaTesi=new Tesi(unTitolo);
  tesi.add(nuovaTesi);
 }
/**Aggiunge uno studente senza nessuna tesi assegnata.
 @param unaMatricola Numero di matricola.*/
 public void aggiungiStudente(int unaMatricola)
 {
  if(cercaStudente(unaMatricola)!=-1)
    return;
  Studente nuovoStudente=new Studente(unaMatricola);
  studenti.add(nuovoStudente);}
```

```
/**Restituisce true se la tesi indicata è libera.
 @param unTitolo Titolo della tesi.
 @return True se la tesi è libera, false altrimenti.*/
 public boolean tesiLibera(String unTitolo)
 {
  if(cercaTesi(unTitolo)==-1)
    return false;
  boolean result=true;
  for(int i=0;i<studenti.size();i++)
  {
   if(studenti.get(i).getTesi()!=null)
     if(studenti.get(i).getTesi().getTitolo().equals(unTitolo))
       result=false;
  }
  return result;
 }
/**Assegna la tesi a uno studente, solo se questa è disponibile.
 @param unaMatricola Numero di matricola.
 @param unTitolo Titolo della tesi.
 @return True se la tesi viene assegnata, false altrimenti.*/
 public boolean assegnaTesi(int unaMatricola, String unTitolo)
 {
  if(cercaStudente(unaMatricola)==-1||cercaTesi(unTitolo)==-1)
    return false;
  if(tesiLibera(unTitolo)==true)
  {
studenti.get(cercaStudente(unaMatricola)).sostituisciTesi(tesi.get(cercaTesi(unT
itolo)));
   return true;
  }
  return false;
 }
/**Rimuove dal sistema uno studente e la tesi a lui assegnata.
 @param unaMatricola Numero di matricola.*/
 public void laureato(int unaMatricola)
 {
  if(cercaStudente(unaMatricola)==-1)
    return;
  String titolo=studenti.get(cercaStudente(unaMatricola)).getTesi().getTitolo();
  tesi.remove(cercaTesi(titolo));
  studenti.remove(cercaStudente(unaMatricola));
 }
/**Assegna una nuova tesi ad uno studente e libera la vecchia tesi.
 @unaMatricola Numero di matricola.
 @titoloNuovaTesi Titolo della nuova tesi.*/
 public void cambioTesi(int unaMatricola, String titoloNuovaTesi)
 {
  if(cercaStudente(unaMatricola)==-1|cercaTesi(titoloNuovaTesi)==-1)
    return;
studenti.get(cercaStudente(unaMatricola)).sostituisciTesi(tesi.get(cercaTesi(tit
oloNuovaTesi)));
 }
/**Restituisce la percentuale di tesi non ancora assegnate presenti nel sistema.
 @return percgentuale di tesi libere.*/
```

```
public int statistica()
 {
  int libere=0;
  for(int i=0;i<tesi.size();i++)
  {
   if(tesiLibera(tesi.get(i).getTitolo())==true)
     libere++;
  }
  int result=(libere*100)/tesi.size();
  return result;
 }
}



public class AssegnazioneTesiTester
{
 public static void main(String[]  args)
 {
  AssegnazioneTesi laurea=new AssegnazioneTesi();
  laurea.aggiungiTesi("Il rating delle obbligazioni.");
  laurea.aggiungiTesi("Facility layout: approcci risolutivi.");
  laurea.aggiungiTesi("L'implementazione della lean production nelle PMI
italiane.");
  laurea.aggiungiTesi("Idrogeno: il futuro delle quattro ruote.");
  laurea.aggiungiStudente(056464);
  laurea.aggiungiStudente(154789);
  laurea.aggiungiStudente(279746);
  System.out.println(laurea.assegnaTesi(056464,"Facility layout: approcci
risolutivi."));
  System.out.println(laurea.tesiLibera("Facility layout: approcci
risolutivi."));
  laurea.laureato(056464);
  System.out.println(laurea.assegnaTesi(154789,"Il rating delle
obbligazioni."));
  System.out.println(laurea.assegnaTesi(279746,"L'implementazione della lean
production nelle PMI italiane."));
  laurea.cambioTesi(279746,"Idrogeno: il futuro delle quattro ruote.");
  System.out.println(laurea.tesiLibera("Il rating delle obbligazioni."));
  System.out.println(laurea.tesiLibera("L'implementazione della lean production
nelle PMI italiane."));
  System.out.println(laurea.statistica());
 }
}
```

Libreria-01/07/16

Testo:

Una negozio di libri desidera gestire un archivio elettronico per tenere traccia delle
spese dei propri clienti. Si assuma una classe Cliente, avente le variabili d'istanza codice (stringa),
spesaEuro (intero) e spesaCentesimi (intero). Assumere gli usuali metodi di accesso e modifica per
tale classe senza svilupparne il codice. La classe Libreria `e cos`ı definita:

```
public class Libreria {
private ArrayList<Cliente> clienti;
public Libreria(){...};
public void aggiungiCliente(String unCodice){...};
public void rimuoviCliente(String unCodice){...};
public void spesaCliente(String unCodice, int unaSpesaEuro,
int unaSpesaCentesimi){...};
public ArrayList<Cliente> premio(int unaSogliaEuro){...};
public void rimuoviListaClienti(ArrayList<Cliente> lista){...};
```

```
public double statistica(){...};
};
```

Il metodo aggiungiCliente aggiunge un nuovo cliente senza creare duplicati. Il metodo rimuoviCliente
rimuove un cliente. Il metodo spesaCliente aggiorna la spesa del cliente per il quantitativo di euro e di
centesimi di euro indicato; nel risultato ottenuto, convertire i centesimi in euro quando questa quantit`a
supera 99. Il metodo premio restituisce la lista dei clienti che hanno accumulato una spesa pari o superiore alla soglia in euro indicata, ed azzera la spesa di questi clienti. Il metodo rimuoviListaClienti
rimuove dall'archivio tutti i clienti presenti nella lista specificata come parametro esplicito. Il metodo
statistica calcola e restituisce la spesa media per cliente.

Consigli:

Attenzione al metodo spesaCliente che traduce i centesimi in euro, per farlo è sufficiente dividere per 99 i centesimi.

```
public class Cliente
{
 private String codice;
 private int spesaEuro;
 private int spesaCentesimi;
    /**Costruisce un oggetto della classe Cliente.
     @param unCodice codice cliente.
     @param euro spesa in euro.
     @param centesimi spesa in centesimi.*/
    public Cliente(String unCodice, int euro, int centesimi)
    {
      codice=unCodice;
      spesaEuro=euro;
      spesaCentesimi=centesimi;
    }
    /**Restituisce il codice cliente.
     @return codice cliente.*/
    public String getCodice()
    {
     return codice;
    }
    /**Restituisce la spesa in euro.
     @return euro spesi.*/
    public int getEuro()
    {
      return spesaEuro;
    }
    /**Restituisce la spesa in centesimi.
     @return cet spesi.*/
    public int getCentesimi()
    {
      return spesaCentesimi;
    }
    /**Modifica la spesa del cliente.
     @param euro nuova cifra.
     @param cent nuova cifra.*/
    public void setSpesa(int euro, int cent)
    {
      spesaEuro=euro;
      spesaCentesimi=cent;
    }
}
```

```
import java.util.ArrayList;
public class Libreria
{
  private ArrayList<Cliente> clienti;
    /**Costruisce un oggetto della classe Libreria.*/
    public Libreria()
    {
      clienti=new ArrayList<Cliente>();
    }
    /**Metodo ausiliario che cerca un cliente nella lista clienti.
     @param unCodice codice cliente.
     @return indice.*/
    private int cercaCliente(String unCodice)
    {
      int result=-1;
      for(int i=0;i<clienti.size();i++)
      {
            if(clienti.get(i).getCodice().equals(unCodice))
                  result=i;
      }
    return result;
    }

/**Aggiunge un nuovo cliente senza creare duplicati.
 @param unCodice codice cliente.*/
public void aggiungiCliente(String unCodice)
{
      if(cercaCliente(unCodice)!=-1)
            return;
      clienti.add(new Cliente(unCodice,0,0));
}
/**Rimuove un cliente da sistema.
 @param unCodice codice cliente.*/
public void rimuoviCliente(String unCodice)
{
      if(cercaCliente(unCodice)==-1)
            return;
            clienti.remove(cercaCliente(unCodice));
}
/**Aggiorna la spesa del cliente per il quantitativo di euro e di
centesimi di euro indicato.
@param unCodice codice cliente.
@param unaSpesaEuro spesa in euro.
@param unaSpesaCentesimi spesa in centesimi.*/
public void spesaCliente(String unCodice, int unaSpesaEuro, int
unaSpesaCentesimi)
{
      if(cercaCliente(unCodice)==-1)
            return;
    int euro=clienti.get(cercaCliente(unCodice)).getEuro()+unaSpesaEuro;
    int
cent=clienti.get(cercaCliente(unCodice)).getCentesimi()+unaSpesaCentesimi;
    int q=cent/99;
    euro+=q;
    cent-=(q*99);
    clienti.get(cercaCliente(unCodice)).setSpesa(euro,cent);
}
/**Restituisce la lista dei clienti che hanno accumulato una spesa pari o
superiore alla soglia in euro indicata.
 @param unaSoglia soglia minima.
 @return lista di clienti.*/
```

```
public ArrayList<Cliente> premio(int unaSoglia)
{
      ArrayList<Cliente> a=new ArrayList<Cliente>();
      for(int i=0;i<clienti.size();i++)
      {
        if(clienti.get(i).getEuro()>=unaSoglia)
        {
            clienti.get(i).setSpesa(0,0);
            a.add(clienti.get(i));
        }
    }
return a;
}
/**Rimuove dall'archivio tutti i clienti presenti nella lista specificata.
 @param lista lista di clienti da cancellare.*/
public void rimuoviListaClienti(ArrayList<Cliente> lista)
{
 for(int i=0;i<lista.size();i++)
      for(int j=clienti.size()-1;j>=0;i--)
      {
      if(lista.get(i)==clienti.get(j))
            clienti.remove(j);
      }
}
/**Calcola e restituisce la spesa media per cliente.
 @return spesa media.*/
public double statistica()
{
 int tot=0;
 for(int i=0;i<clienti.size();i++)
 {
      tot+=clienti.get(i).getEuro();
 }
  return tot/clienti.size();
 }

}



public class LibreriaTester
{
 public static void main(String[] args)
  {
   Libreria giunti=new Libreria();
   giunti.aggiungiCliente("1234");
   giunti.aggiungiCliente("5678");
   giunti.aggiungiCliente("4374");
   giunti.aggiungiCliente("9807");
   giunti.aggiungiCliente("3946");
   giunti.spesaCliente("1234",5,65);
   giunti.spesaCliente("1234",5,80);
   giunti.spesaCliente("5678",15,0);
   System.out.println(giunti.premio(15));
   giunti.rimuoviCliente("9807");
   System.out.println(giunti.statistica());
   }
}
```

Facebook-17/02/10

Testo:

Facebook ha deciso di passare a Java come linguaggio per la sua piattaforma. Nel compito di oggi vi viene richiesto di realizzare una parte del codice. In Facebook, ogni utente è collegato ad un insieme di amici. L'utente può inoltre caricare delle foto ed indicare quali amici sono presenti in ogni foto. In questo esercizio svilupperete una versione semplificata che collega ("tagga") un solo amico ad ogni foto. Si consideri la classe Amici, avente come variabili d'istanza nome (stringa), cognome (stringa) e età (intero). Si consideri inoltre la classe Foto, avente come variabili d'istanza titolo (string) e amico, che rappresenta un riferimento a un oggetto di classe Amico. Si assuma siano già disponibili gli usuali metodi costruttori, accessori e modificatori per ciascuna delle due classi menzionate. Sviluppare tutti i metodi della seguente classe.

```
public class Tag {
private ArrayList<Foto> fotografie;
private Amico[] amici;
private int numAmici;
private final int MAX_NUM_AMICI = 200;
public Tag(){...}
public void aggFoto(String unTitolo){...}
public void aggAmico(String unNome, String unCognome, unaEta){...}
public void aggTag(String unTitolo, String unNomeAmico, String
unCognomeAmico){...}
public void rimuoviAmico(String unNomeAmico, String unCognomeAmico){..}
public double calcolaEtaMedia(){...}
public int contaFotoAmico(String unNome, String unCognome){...}
public Amico calcolaMaxApparizioni() {...}
}
```

Le variabili d'istanza fotografie e amici contengono rispettivamente tutti gli amici e le foto presenti.

La variabile amici è un array di dimensione MAX NUM AMICI, e deve essere gestito come un array parzialmente riempito. Il metodo aggAmico inserisce un nuovo amico (se non esiste già e se vi e disponibilità di spazio). Il metodo aggFoto inserisce una nuova foto (se non esiste già) indicando il titolo che la identifica univocamente e senza alcun amico taggato. Il metodo aggTag inserisce un collegamento tra la foto indicata e l'amico indicato. Se la foto non esiste, il metodo esce senza fare nulla. Se l'amico non esiste deve crearlo, ponendo l'età al valore -1, che serve ad indicare una età sconosciuta. Se non vi è disponibilità di spazio per un nuovo amico, il metodo esce senza fare nulla. Il metodo rimuoviAmico rimuove l'amico indicato assieme a tutte le foto in cui egli è presente. Il metodo calcolaEtaMedia calcola l'età media degli amici, non considerando gli amici di cui non si conosce l'età. Il metodo contaFotoAmico calcola il numero di foto in cui appare l'amico indicato. Il metodo calcolaMaxApparizioni restituisce un riferimento ad un oggetto di classe Amico presente nel maggior numero di foto (risolvere arbitrariamente i casi di parità).

Consigli:

Si invita ad implementare due metodi ausiliari che verifichino la presenza di un amico e di una foto rispettivamente nell'Array amici e nell'ArrayList fotografie. Il metodo rimuoviAmico può dare difficoltà in quanto non solo rimuove l'amico dall'Array amici, ma rimuove anche tutte le foto in cui l'amico da eliminare è taggato(si consiglia di rivedere l'esercizio 7.1 poiché le foto da eliminare possono essere contigue nell'ArrayList fotografie).

Nella classe Foto si consiglia di implementare un metodo setAmico che aiuta la stesura della classe Tag.

```
public class Amici
{
private String nome;
private String cognome;
private int eta;
/**Creo un oggetto della classe Amici.*/
 public Amici(String unNome, String unCognome, int unEta)
 {
  nome = unNome;
  cognome = unCognome;
  eta = unEta;
 }
/**Restituisce il nome dell'amico.
 @return Nome dell'amico.*/
 public String getNome()
 {
  return nome;
 }
/**Restituisce il cognome dell'amico.
 @return Cognome dell'amico.*/
 public String getCognome()
 {
  return cognome;
 }
/**Restituisce l'età dell'amico.
 @return Età dell'amico.*/
 public int getEta()
 {
  return eta;
 }
}
```

```
public class Foto
{
private String titolo;
private Amici amico;
/**Creo un oggetto della classe Foto.
 @unTitolo Titolo foto.
 @unAmico Un oggetto della classe Amico.*/
 public Foto(String unTitolo, Amici unAmico)
 {
  titolo= unTitolo;
  amico= unAmico;
 }
/**Cambio l'amico presente in una foto.
 @param unAmico Amico da assegnare ad una foto.*/
 public void setAmico(Amici unAmico)
 {
  amico= unAmico;
 }
/**Restituisce il titolo della foto.
 @return Il titolo della foto.*/
 public String getTitolo()
 {
  return titolo;
 }
/**Restituisce l'amico assegnato ad una foto.
 @return L'amico assegnato alla foto.*/
```

```
public Amici getAmico()
 {
  return amico;
 }
}


import java.util.Arrays;
import java.util.ArrayList;
public class Tag
{
private ArrayList<Foto> fotografie;
private Amici[] amici;
private int numAmici;
private final int MAX_NUM_AMICI = 200;
/**Creo un oggetto della classe Tag.*/
 public Tag()
 {
  fotografie= new ArrayList<Foto>();
  amici= new Amici[MAX_NUM_AMICI];
  numAmici=0;
 }
/**Metodo ausiliario che riestituisce l'indice corrispondente all'amico cercato.
 @param unNome Nome dell'amico.
 @param unCognome Cognome dell'amico.
 @return Indice dell'amico.*/
 private int cercaAmico(String unNome, String unCognome)
 {
  int result=-1;
  for(int i=0; i<numAmici; i++)
  {
  if(amici[i].getNome().equals(unNome)&&amici[i].getCognome().equals(unCognome))
    result=i;
  }
  return result;
 }
/**Metodo ausiliario che riestituisce l'indice corrispondente alla foto cercata.
 @param unTitolo Titolo della foto.
 @return Indice della foto.*/
 private int cercaFoto(String unTitolo)
 {
  int result=-1;
  for(int i=0; i<fotografie.size(); i++)
  {
  if(fotografie.get(i).getTitolo().equals(unTitolo))
    result=i;
  }
  return result;
 }

/**Aggiungi un amico all'Array amici.
 @param unNome Nome dell'amico.
 @param unCognome Cognome dell'amico.
 @param unEta Età dell'amico.*/
 public void aggAmico(String unNome, String unCognome, int unEta)
 {
  if(cercaAmico(unNome, unCognome)!=-1|numAmici>MAX_NUM_AMICI)
    return;
  if(numAmici< MAX_NUM_AMICI)
  {
  amici[numAmici]= new Amici(unNome, unCognome, unEta);
  numAmici++;
  }}
```

```
/**Aggiunge una foto all'ArrayList fotografie.
 @param unTitolo Titolo della fotografia.*/
 public void aggFoto(String unTitolo)
 {
  if(cercaFoto(unTitolo)!=-1)
    return;
  Foto nuovaFoto=new Foto(unTitolo, null);
  fotografie.add(nuovaFoto);
 }
/**Tagga un amico in una foto.
 @param unTitolo Titolo della foto.
 @param unNomeAmico Nome di un amico.
 @param unCognomeAmico Cognome di un amico.*/
 public void aggTag(String unTitolo, String unNomeAmico, String unCognomeAmico)
 {
  if(cercaFoto(unTitolo)==-1||numAmici> MAX_NUM_AMICI)
    return;
  else if(cercaAmico(unNomeAmico,unCognomeAmico)!=-1)
    fotografie.get(cercaFoto(unTitolo)).setAmico(amici[cercaAmico(unNomeAmico,
unCognomeAmico)]);
  else if(cercaAmico(unNomeAmico, unCognomeAmico)==-1)
  {
   if(numAmici>MAX_NUM_AMICI)
     return;
   aggAmico(unNomeAmico, unCognomeAmico, -1);
   fotografie.get(cercaFoto(unTitolo)).setAmico(amici[cercaAmico(unNomeAmico,
unCognomeAmico)]);
  }
 }
/**Rimuove l'amico dall'Array amici e rimuove tutte le foto dove l'amico è
taggato.
 @param unNomeAmico Nome di un amico.
 @param unCognomeAmico Cognome di un amico.*/
 public void rimuoviAmico(String unNomeAmico, String unCognomeAmico)
 {
  if(cercaAmico(unNomeAmico,unCognomeAmico)==-1)
    return;
  for(int i=fotografie.size()-1; i>-1; i--)
  {
if(fotografie.get(i).getAmico().getNome().equals(unNomeAmico)&&fotografie.get(i)
.getAmico().getCognome().equals(unCognomeAmico))
   fotografie.remove(i);
  }
  amici[cercaAmico(unNomeAmico,unCognomeAmico)]=amici[numAmici-1];
  numAmici--;
 }
/**Restituisce il valore dell'età media degli amici,
 non tenendo conto di quelli di cui non si conosce L'età.
 @return Età media amici.*/
 public double calcolaEtaMedia()
 {
  double etaTotale=0;
  int cont=0;
  for(int i=0; i< numAmici; i++)
  {
   if(amici[i].getEta()!=-1)
   {
    etaTotale= etaTotale + amici[i].getEta();
    cont++;
   }
  }
  double etaMedia= etaTotale/cont;
  return etaMedia;}
```

```
/**Restituisce il numero di foto in cui appare un amico.
 @param unNome Nome dell'amico.
 @param unCognome Cognome dell'amico.
 @return Numero di foto in cui appare un amico.*/
 public int contaFotoAmico(String unNome, String unCognome)
 {
  int cont=0;
  if(cercaAmico(unNome, unCognome)==-1)
    return -1;
  for(int i=0; i<fotografie.size(); i++)
  {
 if(fotografie.get(i).getAmico().getNome().equals(unNome)&&fotografie.get(i).get
Amico().getCognome().equals(unCognome))
  cont++;
  }
  return cont;
 }
/**Restituisce un oggetto della classe Amici che compare il maggior numero di
volte nelle foto.
 @return Amico che compare più volte nelle foto.*/
 public Amici calcolaMaxApparizioni()
 {
  int fotoVuota=0;
  int numMax=0;
  int numIndex=-1;
  for(int i=0;i<numAmici;i++)
  {
   int comparsa=0;
   for(int j=0;j<fotografie.size();j++)
   {
    if(fotografie.get(j).getAmico()==null)
      fotoVuota++;
    else if(fotografie.get(j).getAmico().getNome().equals(amici[i].getNome()))
      comparsa++;
   }
   if(comparsa>numMax)
   {
   numMax=comparsa;
   numIndex=i;
   }
  }
  return amici[numIndex];
 }
}



public class TagTester
{
 public static void main(String[] args)
 {
  Tag album = new Tag();
  album.aggAmico("Raissa","Pizzato",19);
  album.aggAmico("Marco","Zorzi",22);
  album.aggAmico("Michele","Duregon",27);
  album.aggFoto("Laurea");
  album.aggFoto("Mare");
  album.aggFoto("Scuola");
  album.aggFoto("Audi");
  album.aggTag("Laurea","Raissa","Pizzato");
  album.aggTag("Mare","Michele","Duregon");
  album.aggTag("Scuola","Marco","Zorzi");
  album.aggTag("Audi","Raissa","Pizzato");
```

```
  System.out.println(album.calcolaEtaMedia());
  System.out.println(album.contaFotoAmico("Raissa","Pizzato"));
  System.out.println(album.contaFotoAmico("Marco","Zorzi"));
  System.out.println(album.contaFotoAmico("Michele","Duregon"));
  System.out.println(album.calcolaMaxApparizioni());
 }
}
```

Agenzia-27/02/16

Testo:

Una agenzia turistica fornisce appartamenti in affitto ai propri clienti. Si assuma una classe Appartamento, avente le variabili d'istanza citta (stringa) e prezzo (intero), ed una classe Cliente, avente la variabile d'istanza cognome (stringa) e unAppartamento (riferimento a Appartamento). Si assumano gli usuali metodi di accesso e modifica per tali classi (non sviluppare il codice). La classe Agenzia `e cos`ı definita:

```
public class Agenzia {
private Appartamento[] appartamenti;
private int numAppartamenti;
private ArrayList<Cliente> clienti;
public Agenzia(int maxAppartamenti){};
public void aggCliente(String unCognome);
public boolean aggAppartamento(String unaCitta, int unPrezzo);
public boolean arrivo(String unCognome, String unaCitta, int maxPrezzo);
public void partenza(String unCognome);
public void rimuoviAppartamento(int indice);
public ArrayList<Appartamento> lista(String unaCitta);
public String statistica();
};
```

Gli appartamenti sono organizzati in un array parzialmente riempito. Un appartamento `e libero se non `e assegnato ad alcun cliente. Il metodo aggCliente aggiunge un nuovo cliente senza creare duplicati.
Il metodo aggAppartamento aggiunge un nuovo appartamento, e restituisce false se l'operazione non ha successo. Il metodo arrivo assegna al cliente indicato un appartamento libero nella citt`a specificata e con prezzo non superiore a quello indicato; restitire false se non vi sono appartamenti disponibili a quel prezzo. Il metodo partenza libera l'appartamento occupato dal cliente indicato, senza rimuovere il cliente dalla lista dei clienti. Il metodo rimuoviAppartamento rimuove dal sistema l'appartamento avente l'indice specificato solamente se questo `e libero. Il metodo lista restituisce una lista di appartamenti nella città indicata. Il metodo statistica restituisce la citt`a avente il maggior numero di appartamenti (risolvere arbitrariamente i casi di parità).

Consigli:

In questo esercizio è particolarmente utile creare un metodo ausiliario boolean che controlla se l'appartamento è libero.

```
public class Appartamento
{
 private String citta;
 private int prezzo;
```

```
/**Costruisce un oggetto della classe Appartamento.
 @param unaCitta città.
 @param unPrezzo prezzo.*/
 public Appartamento(String unaCitta, int unPrezzo)
 {
 citta=unaCitta;
 prezzo=unPrezzo;
 }
 /**Restituisce la città.
  @return città.*/
 public String getCitta()
 {
 return citta;
 }
 /**Restituisce il prezzo.
   @return prezzo.*/
 public int getPrezzo()
 {
 return prezzo;
 }
}


public class Cliente
{
private String cognome;
private Appartamento appartamento;
/**Costruisce un oggetto della classe Cliente.
 @param unCognome cognome cliente.*/
public Cliente(String unCognome)
{
cognome=unCognome;
appartamento=null;
}
/**Restituisce il cognome del cliente.
 @return cognome.*/
public String getCognome()
{
return cognome;
}
/**Restituisce l'appartamento associato al cliente.
 @return appartamento.*/
public Appartamento getAppartamento()
{
return appartamento;
}
/**Modifica l'appartamento associato al cliente.
 @param nuovo appartamento.*/
public void setAppartamento(Appartamento nuovo)
{
appartamento=nuovo;
}
}


import java.util.ArrayList;
public class Agenzia
{
private Appartamento[] appartamenti;
private int numAppartamenti;
private ArrayList<Cliente> clienti;
```

```
/**Costruisce un oggetto di classe Agenzia.
 @param max numero appartamenti.*/
public Agenzia(int max)
{
appartamenti= new Appartamento [max];
clienti=new ArrayList<Cliente>();
numAppartamenti=0;
}
/**Metodo ausiliario che controlla se un appartamento è libero.
 @param indiceApp indice appartamento.
 @return true o false.*/
private boolean apLibero(int indiceApp)
{
boolean result=true;
for(int i=0;i<clienti.size();i++)
{
if(clienti.get(i).getAppartamento()==appartamenti[indiceApp])
  result=false;
}
return result;
}
/**Metodo ausiliario che cerca un cliente.
 @param unCognome cognome cliente.
 @return indice.*/
private int cercaCliente(String unCognome)
{
int result=-1;
for(int i=0;i<clienti.size();i++)
{
if(clienti.get(i).getCognome().equals(unCognome))
result=i;
}
return result;
}
/**Aggiunge un nuovo cliente.
 @param unCognome cognome cliente.*/
public void aggCliente(String unCognome)
{
if(cercaCliente(unCognome)!=-1)
return;
clienti.add(new Cliente(unCognome));
}
/**Aggiunge un nuovo appartamento.
 @param unaCitta città.
 @param unPrezzo prezzo.
 @return true o false.*/
public boolean aggAppartamento(String unaCitta, int unPrezzo)
{
if(numAppartamenti>=appartamenti.length)
return false;
appartamenti[numAppartamenti]=new Appartamento(unaCitta, unPrezzo);
numAppartamenti++;
return true;
}
/**Assegna al cliente indicato un appartamento libero nella città
specificata e con prezzo non superiore a quello indicato.
@param unCognome cognome.
@param unaCitta città.
@param maxPrezzo prezzo massimo.*/
```

```
public boolean arrivo(String unCognome, String unaCitta,int maxPrezzo)
{
if(cercaCliente(unCognome)==-1)
  return false;
  for(int i=0;i<numAppartamenti;i++)
  {
  if(appartamenti[i].getCitta().equals(unaCitta))
    if(appartamenti[i].getPrezzo()<=maxPrezzo)
       if(apLibero(i)==true)
     {
       clienti.get(cercaCliente(unCognome)).setAppartamento(appartamenti[i]);
       return true;
     }
  }
  return true;
 }
/**Libera l'appartamento occupato dal cliente indicato.
 @param unCognome cognome.*/
 public void partenza(String unCognome)
 {
  clienti.get(cercaCliente(unCognome)).setAppartamento(null);
 }
/** Rimuove dal sistema
l'appartamento avente l'indice specificato solamente se questo è libero.
@param indice indice.*/
public void rimuoviAppartamento(int indice)
 {
  if(indice>=numAppartamenti||apLibero(indice)==false)
      return;
  appartamenti[indice]=appartamenti[numAppartamenti-1];
  numAppartamenti--;
 }

/**Restituisce una lista di appartamenti nella città indicata.
 @param unaCitta città.
 @return lista appartamenti.*/
public ArrayList<Appartamento> lista(String unaCitta)
{
 ArrayList<Appartamento> city=new ArrayList<Appartamento>();
 for(int i=0;i<numAppartamenti;i++)
 {
  if(appartamenti[i].getCitta().equals(unaCitta))
      city.add(appartamenti[i]);
 }
return city;
}
/**restituisce la città avente il maggior numero di appartamenti.
 @return città.*/
```

```
public String statistica()
{
int indice=-1;
int max=0;
for(int i=0;i<numAppartamenti;i++)
{
 int cont=0;
   for(int j=0;j<numAppartamenti;j++)
   {
    if(appartamenti[i].getCitta().equals(appartamenti[j].getCitta()))
        cont++;
    if(j==numAppartamenti-1&&cont>max)
    {
     max=cont;
     indice=i;
    }
   }
 }
 return appartamenti[indice].getCitta();
}

}



public class AgenziaTester
{
  public static void main(String[] args)
      {
     Agenzia tecnocasa=new Agenzia(5);
     tecnocasa.aggCliente("Pizzato");
     tecnocasa.aggCliente("Zorzi");
     tecnocasa.aggCliente("Bressan");
     tecnocasa.aggCliente("Fioretto");
     System.out.println("Inserisco appartamenti:");
     System.out.println(tecnocasa.aggAppartamento("Vicenza",100));
     System.out.println(tecnocasa.aggAppartamento("Venezia",100));
     System.out.println(tecnocasa.aggAppartamento("Vicenza",200));
     System.out.println(tecnocasa.aggAppartamento("Verona",600));
     System.out.println("Inserisco clienti:");
     System.out.println(tecnocasa.arrivo("Zorzi","Vicenza",100));
     System.out.println(tecnocasa.arrivo("Pizzato","Venezia",450));
     tecnocasa.partenza("Pizzato");
     System.out.println(tecnocasa.lista("Vicenza"));
     tecnocasa.rimuoviAppartamento(3);
     System.out.println(tecnocasa.lista("Vicenza"));
     System.out.println(tecnocasa.statistica());
   }
}
```

Trasporti-31/01/09

Testo:

Una società di trasporti vuole gestire elettronicamente l'allocazione dei propri automezzi sulle destinazioni servite. Si sviluppi la classe Destinazione, avente come variabili d'istanza citta (stringa) e distanza (int), e la classe Automezzo, avente come variabili d'istanza targa (stringa) e arrivo (riferimento a Destinazione). Si assumano gli usuali metodi costruttori, accessori e modifi catori. Sviluappare la seguente classe:

```
public class Trasporti {
private ArrayList<Automezzo> mezzi; // lista di automezzi
private Destinazione[] destinazioni; // array di destinazioni servite
private int nDestinazioni; // numero di destinazioni servite
```

```
public Trasporti(int max){...} // max = numero massimo di destinazioni servite
public void aggiungiDestinazione(String unaCitta, int unaDistanza){...}
public void aggiungiMezzo(String unaTarga, String unArrivo){...}
public void terminaViaggio(String unaTarga){...}
public void assegnaDestinazione(String unaTarga, String unaCitta){...}
public void eliminaDestinazione(String unaCitta){...}
public int stat(){...}
}
```

Il costruttore inizializza un oggetto con un array destinazioni inizialmente vuoto ed avente dimensione specificata dal parametro esplicito. I metodi di inserimento aggiungono oggetti solamente se questi non sono già presenti (suggerimento: usare appropriati metodi di ricerca per testare questa condizione). Inoltre, aggiungiMezzo crea la nuova destinazione associata alla città di arrivo, se questa non è già presente. Il metodo terminaViaggio libera l'automezzo specificato dal parametro esplicito (un automezzo è libero se non ha alcuna destinazione assegnata). Il metodo assegnaDestinazione assegna ad un automezzo una destinazione. Il metodo eliminaDestinazione rimuove la destinazione specificata, e libera tutti gli automezzi associati a tale destinazione. Infine, il metodo stat restituisce la media delle distanze che ciascun automezzo dovrà percorrere.

Consigli:
Si invita ad implementare due metodi ausiliari che verifichino la presenza di un targa e di una destinazione rispettivamente nell'ArrayList mezzi e nell'Array destinazioni. Si consiglia inoltre di implementare un metodo setDestinazione nella classe Automezzo in modo da semplificare la stesura della classe Trasporti.

```
public class Destinazione
{
private String città;
private int distanza;
/**Creo un oggetto della classe Destinazione.
 @param unaCittà Città.
 @param unaDistanza Distanza.*/
 public Destinazione(String unaCittà, int unaDistanza)
 {
  città = unaCittà;
  distanza = unaDistanza;
 }
/**Creo un oggetto della classe Destinazione,
  con distanza uguale a zero.
 @param unaCittà Città.*/
 public Destinazione(String unaCittà)
 {
  città = unaCittà;
  distanza = 0;
 }
/**Restituisce la città.
 @return Città.*/
 public String getCittà()
 {
  return città;
 }
/**Restituisce la distanza.
 @return Distanza.*/
 public int getDistanza()
 {
  return distanza;
 }
/**Restituisce città più distanza.
 @return Città più distanza.*/
```

```
 public String toString()
 {
  return città+distanza;
 }
}



public class Automezzo
{
private String targa;
private Destinazione arrivo;
/**Crea un oggetto della classe Automezzo.
 @param unaTarga Targa del veicolo.
 @param unArrivo Oggetto della classe Destinazione.*/
 public Automezzo(String unaTarga, Destinazione unArrivo)
 {
  targa = unaTarga;
  arrivo = unArrivo;
 }
/**Restituisce la targa dell'automezzo.
 @return Targa automezzo.*/
 public String getTarga()
 {
  return targa;
 }
/**Restituisce la destinazione dell'automezzo.
@return Destinazione automezzo.*/
 public Destinazione getDestinazione()
 {
  return arrivo;
 }
/**Modifica la destinazione a un automezzo.
 @param unaDestinazione Nuova destinazione.*/
 public void setArrivo(Destinazione unaDestinazione)
 {
  arrivo = unaDestinazione;
 }
/**Restituisce targa più arrivo.
 @return targa più arrivo.*/
 public String toString()
 {
  return targa+arrivo;
 }
}



import java.util.Arrays;
import java.util.ArrayList;
public class Trasporti
{
private ArrayList<Automezzo> mezzi;
private Destinazione[] destinazioni;
private int nDestinazioni;
/**Crea un oggetto della classe Trasporti.
 @param max Dimensione array destinazioni.*/
 public Trasporti(int max)
 {
  destinazioni = new Destinazione[max];
  nDestinazioni = 0;
  mezzi = new ArrayList<Automezzo>();
 }
```

```
/**Primo metodo ausiliario, cerca un automezzo con la targa specificata.
 @param unaTarga Targa dell'automezzo da cercare.
 @return result Indice targa.*/
 private int cercaTarga(String unaTarga)
 {
  int result=-1;
  for(int i = 0; i < mezzi.size(); i++)
  {
   if(mezzi.get(i).getTarga().equals(unaTarga))
     result=i;
  }
  return result;
 }
/**Secondo metodo ausiliario, cerca una destinazione con città specificata.
 @param unaCittà Città della destinazione da cercare.
 @return result Indice città.*/
 private int cercaCittà(String unaCittà)
 {
  int result=-1;
  for(int i = 0; i < nDestinazioni; i++)
  {
   if(destinazioni[i].getCittà().equals(unaCittà))
     result=i;
  }
  return result;
 }
/**Aggiunge un oggetto Destinazione all'array destinazioni.
 @param unaCittà Città.
 @param unaDistanza Distanza.*/
 public void aggiungiDestinazione(String unaCittà, int unaDistanza)
 {
  if(cercaCittà(unaCittà)!=-1|nDestinazioni>=destinazioni.length)
    return;
  Destinazione unaDestinazione = new Destinazione(unaCittà,unaDistanza);
destinazioni[nDestinazioni] = unaDestinazione;
  nDestinazioni++;
}
/**Aggiunge un mezzo all'ArrayList mezzi, se la destinazione non è già presente,
 la aggiunge all'Array destinazioni.
 @param unaTarga Targa automezzo.
 @param unArrivo Città di arrivo.*/
 public void aggiungiMezzo(String unaTarga, String unArrivo)
 {
  if(cercaTarga(unaTarga)!=-1)
    return;
  if(cercaCittà(unArrivo)!=-1)
  {
   Automezzo c = new Automezzo(unaTarga,destinazioni[cercaCittà(unArrivo)]);
   mezzi.add(c);
  }
  if(cercaCittà(unArrivo)==-1)
  {
   aggiungiDestinazione(unArrivo,0);
   Automezzo b = new Automezzo(unaTarga,destinazioni[cercaCittà(unArrivo)]);
   mezzi.add(b);
  }
 }
```

```
/**Libera il mezzo con unaTarga da una destinazione.
 @param unaTarga Traga automezzo da liberare.*/
 public void terminaViaggio(String unaTarga)
 {
  if(cercaTarga(unaTarga)==-1)
    return;
  mezzi.get(cercaTarga(unaTarga)).setArrivo(null);
 }
/**Assegna una destinazione a un veicolo.
 @param unaTarga Traga automezzo.
 @param unaCittà Città della destinazione.*/
 public void assegnaDestinazione(String unaTarga, String unaCittà)
 {
  if(cercaTarga(unaTarga)==-1)
    return;
  mezzi.get(cercaTarga(unaTarga)).setArrivo(new Destinazione(unaCittà));
 }
 /**Elimina una destinazione dall'Array destinazioni.
 @param unaCittà Città della destinazione.*/
 public void eliminaDestinazione(String unaCittà)
 {
  if(cercaCittà(unaCittà)==-1)
    return;
  for(int i=0;i<mezzi.size();i++)
  {
   if(mezzi.get(i).getDestinazione()!=null)
     if(mezzi.get(i).getDestinazione().getCittà().equals(unaCittà))
       mezzi.get(i).setArrivo(null);
  }
  destinazioni[cercaCittà(unaCittà)] = destinazioni[nDestinazioni-1];
  nDestinazioni--;
 }
/**Restituisce la media distanza che ciascun automezzo dovrà percorrere.
 @return media Distanza media.*/
 public int stat()
 {
  int totale = 0;
  for(int i=0;i<mezzi.size();i++)
  {
   if(mezzi.get(i).getDestinazione()!=null)
     totale = totale + mezzi.get(i).getDestinazione().getDistanza();
  }
  int media = totale/mezzi.size();
  return media;
 }
}



public class TrasportiTester
{
 public static void main(String[] args)
 {
  Trasporti alpeTrans = new Trasporti(50);
  alpeTrans.aggiungiDestinazione("Marostica", 9);
  alpeTrans.aggiungiDestinazione("Vicenza", 36);
  alpeTrans.aggiungiDestinazione("Bassano", 24);
  alpeTrans.aggiungiDestinazione("Trento", 127);
  alpeTrans.aggiungiDestinazione("Conco", 11);
  alpeTrans.aggiungiDestinazione("Asiago", 19);
  alpeTrans.aggiungiMezzo("RA 907 MX","Marostica");
  alpeTrans.aggiungiMezzo("SE 694 UY","Vicenza");
  alpeTrans.aggiungiMezzo("QU 490 PL","Bassano");
```

```
  alpeTrans.aggiungiMezzo("EW 345 JD","Trento");
  alpeTrans.aggiungiMezzo("AY 040 JD","Conco");
  alpeTrans.aggiungiMezzo("AX 883 DB","Asiago");
  alpeTrans.assegnaDestinazione("RA 907 MX","Asiago");
  alpeTrans.terminaViaggio("EW 345 JD");
  alpeTrans.eliminaDestinazione("Trento");
  System.out.println(alpeTrans.stat());
 }
}
```

TEMI D'ESAME SVOLTI (Esercizi 4)

Shuffle

Testo:

Date due stringhe, la stringa shuffle delle due si ottiene alternando i caratteri della prima stringa con i caratteri della seconda, e copiando i rimanenti caratteri nel caso le due stringhe abbiano diversa lunghezza. Ad esempio, date le stringhe "dna" e "on", la stringa shuffle delle due è "donna". Si implementi un metodo statico String shuffle(String s1, String s2){...} che restituisce la stringa shuffle di s1 e s2.

Consigli:

Si consiglia di dividere il metodo in due casi imponendo una condizione che controlli se la stringa s1 è più lunga della stringa s2. Inoltre si ricorda che il metodo substring(int a), con un solo parametro esplicito, copia tutti i caratteri da a(numero passato come parametro)non compreso in poi.

```
public class Shuffle
{
 public static String shuffle(String s1, String s2)
 {
  String r="";
  if(s1.length()>=s2.length())
  {
   for (int i=0;i<s2.length(); i++)
   r=r+s1.substring(i,i+1) + s2.substring(i,i+1);
   r=r+s1.substring(s2.length());
  }
  else
  {
   for(int j=0; j<s1.length();j++)
   r=r+s1.substring(j,j+1) + s2.substring(j,j+1);
   r=r+s2.substring(s1.length());
  }
  return r;
 }
}


import java.util.Scanner;
public class ShuffleTester
{
 public static void main (String[] args)
 {
  Scanner input = new Scanner(System.in);
  System.out.println("Inserisci la prima stringa");
  String s1= input.next();
  System.out.println("Inserisci la seconda stringa");
  String s2= input.next();
  System.out.println(Shuffle.shuffle(s1,s2));
 }
}
```

Bilanciato

Testo:

Un array di interi di lunghezza n si dice bilanciato se esiste un indice i tale che la somma dei primi i elementi dell'array sia uguale alla somma dei rimanenti n – i elementi. Ad esempio, l'array [20, -4, 11, 1, 4] è bilanciato, perchè i primi due elementi hanno somma uguale ai rimanenti elementi, mentre l'array [7, 13, 6, 8] non è bilanciato, perché nessun indice verifica la proprietà richiesta. Sviluppare un metodo public static boolean bilanciato(int[] unArray){...} che restituisce true se unArray è bilanciato, e false altrimenti.

Consigli:

Per risolvere questo esercizio è necessario usare due for innestati. Vanno create 2 variabili int sx(sinistra), dx(destra). Nel primo for si inserisce in sx il primo elemento di unArray e si pone dx=0. Nel secondo for si sommano tutti i restanti elementi, il risultato va inserito in dx. Fuori dal for più interno si verifica se sx e dx sono uguali. Quindi se sono uguali si restituisce true e il metodo finisce, se non sono uguali il compilatore ritorna al primo for e somma a sx anche il secondo elemento di unArray. Nel secondo for si sommano tutti i restanti elementi, il risultato va inserito in dx. si verifica se sx e dx sono uguali ecc. se l'uguaglianza non si verifica mai si restituisce false.

```
public class Bilanciato
{
public static boolean bilanciato(int[] unArray)
{
 int sx=0;
 int dx=0;
 for (int i=0;i<unArray.length;i++)
 {
  sx=sx+unArray[i];
  dx=0;
  for (int j=i+1;j<unArray.length;j++)
   {
   dx=dx+unArray[j];
   }
  if(sx==dx)
    return true;
 }
  return false;
}
}
```

```
public class BilanciatoTester
{
 public static void main(String[] args)
 {
 int[] zuzi={20, -4, 11, 1, 4};
 System.out.println(Bilanciato.bilanciato(zuzi));
 }
}
```

Pari

Testo:

Un array di interi si dice pari se ciascuno dei suoi elementi appare un numero pari di volte. Ad esempio, l'array [20, 1, 1, 20] è pari, mentre l'array [20, 1, 20, 1, 1] non è pari,poichè l'elemento 1 appare un numero dispari di volte. Sviluppare un metodo public static boolean isPari(int[] myArray) che restituisce true se myArray è pari, false altrimenti.

Consigli:

Si utilizzano due for innestati.

```
public class Pari
{
 public static boolean isPari(int[] myArray)
 {
  if(myArray.length%2!=0||myArray.length==0)
    return false;
  for(int i=0; i<myArray.length; i++)
  {
   int c=0;
   for(int j=0; j<myArray.length; j++)
   {
    if(myArray[i]==myArray[j])
      c++;
    if(c%2!=0&&j==myArray.length-1)
      return false;
   }
  }
  return true;
 }
}
```

```
public class PariTester
{
 public static void main(String[] args)
 {
  int[] myArray=new int[4];
  myArray[0]=1;
  myArray[1]=4;
  myArray[2]=1;
  myArray[3]=4;
  System.out.println(Pari.isPari(myArray));
 }
}
```

MyString

Testo:

All'interno di una classe MyString priva di variabili d'istanza, si implementi un metodo public static boolean noDuplicate(String s) che restituisce true se la stringa s non contiene alcun carattere ripetuto, e false altrimenti. Ad esempio, per la stringa "automobile" il metodo restituisce false, perché il carattere 'o' appare più di una volta.

Consigli:

Sono state sviluppate due soluzioni statiche differenti, una con charAt e una con substrinng. In entrambi i casi sono necessari due for innestati.

Soluzione con charAt

```
public class MyString
{
 public static boolean noDuplicate(String s)
 {
  if(s.length()==0)
    return true;
  for(int i=0;i<s.length();i++)
  {
   int app=0;
   for(int j=0;j<s.length();j++)
   {
    if(s.charAt(i)==s.charAt(j))
      app++;
    if(app>1)
      return false;
   }
  }
  return true;
 }
}
```

Soluzione con substring

```
public class MyString
{
 public static boolean noDuplicate(String s)
 {
  String my=s;
  if(my.length()==0)
    return true;
  for(int i=0; i<my.length()-1; i++)
  {
   for(int j=i+1;j<my.length();j++)
   {
    if (my.substring(i,i+1).equals(my.substring(j,j+1)))
      return false;
   }
  }
  return true;
 }
}
```

```
public class MyStringTester
{
 public static void main(String[] args)
 {
  System.out.println(MyString.noDuplicate("audiR8"));
 }
}
```

LaMiaStringa

Testo:

Si assuma una classe LaMiaStringa con una variabile d'istanza private String str. Si implementi un metodo public boolean multiple(int n)f...g che restituisce true se e solo se str contiene un carattere che appare esattamente un numero di volte pari al valore del parametro n. Ad esempio, per la stringa "abracadabra" la chiamata multiple(int 5) restituisce true poichè esiste un carattere che appare esattamente 5 volte, mentre multiple(int 3) restituisce false, poichè nessun carattere appare esattamente 3 volte.

Consigli:

Utilizzare for innestati.

```
public class LaMiaStringa
{
private String stringa;

 public LaMiaStringa(String unaStringa)
 {
  stringa = unaStringa;
 }

 public boolean multiple(int n)
 {
  int cont = 0;
  String sub1="";
  String sub2="";
  for(int i=0; i< stringa.length(); i++)
  {
   cont=0;
   sub1= stringa.substring(i,(i+1));
   for(int j=0; j<stringa.length(); j++)
   {
    sub2= stringa.substring(j, (j+1));
    if(sub1.equalsIgnoreCase(sub2))
      cont++;
    if(cont==n&&j==stringa.length()-1)
      return true;
   }
  }
 return false;
 }
}
```

```
public class LaMiaStringa Tester
{
 public static void main(String[] args)
 {
  MyString rais = new LaMiaStringa("automobile");
  System.out.println(rais.multiple(2));
 }
}
```

ArrayBilanciati

Testo:

Due array di interi si dicono bilanciati se hanno la stessa lunghezza e la somma dei primi elementi è uguale alla somma dei secondi elementi, che è uguale alla somma dei terzi elementi, e così via. Sviluppare un metodo statico boolean bilanciati(int[] a1, int[] a2) che restituisce true se e solo se i due array a1, a2 sono bilanciati.

Consigli:

Si consiglia di controllare per prima cosa che la lunghezza dei due array sia uguale. Si invita successivamente ad usare un ciclo for che confronti la somma dei primi elementi dei due array con la somma dei successivi n elementi dei due array.

```
public class Bilanciato
{
 public static boolean bilanciato(int[]a1, int[]a2)
 {
  if(a1.length != a2.length)
    return false;
  for(int i=1; i<a1.length; i++)
  {
   if(a1[0]+a2[0]!=a1[i]+a2[i])
     return false;
  }
  return true;
 }
}
```

```
public class BilanciatoTester
{
 public static void main(String[] args)
 {
  int[] primo= new int[3];
    primo[0]=1;
    primo[1]=3;
    primo[2]=5;

  int[] secondo= new int[3];
    secondo[0]=8;
    secondo[1]=6;
    secondo[2]=4;
  System.out.println(Bilanciato.bilanciato(primo, secondo));
 }
}
```

Tandem

Testo:

Un tandem è un array di interi di lunghezza pari, composto dalla concatenazione di due array uguali. Ad esempio, l'array [4,7,1,4,7,1] è un tandem, mentre l'array [4,7,1,4,1,7] non è un tandem. Sviluppare un metodo statico boolean isTandem(int[] a) che restituisce true se e solo se l'array a è un tandem.

Consigli:

Si consiglia di controllare per prima cosa che la lunghezza dei due array sia pari. Si invita successivamente ad usare un ciclo for che confronti la prima parte dell'array con la seconda parte dell'array.

```
public class Tandem
{
 public static boolean isTandem(int[] a)
 {
  if(a.length%2 != 0)
    return false;
  for(int i=0;i<a.length/2;i++)
  {
   if(a[i]!=a[i+a.length/2])
     return false;
  }
  return true;
 }
}



public class TandemTester
{
 public static void main(String[] args)
 {
  int[] g = new int[6];
  g[0]=4;
  g[1]=7;
  g[2]=1;
  g[3]=4;
  g[4]=7;
  g[5]=1;
  System.out.println(Tandem.isTandem(g));
 }
}
```

Infix

Testo:

Sviluppare un metodo statico boolean infix(int[] a1, int[] a2) che restituisce true se l'array a1 appare come sottosequenza contigua all'interno dell'array a2, e restituisce false altrimenti.
Ad esempio, [13, -1, 17] appare come sottosequenza contigua in [8, -7, 13, -1, 17, 8] ma non in [8, 13, -7, -1, 17, 8].

Consigli:

Si consiglia di usare un metodo ausiliario che controlla se a1 appare in a2 a partire da una assegnata posizione k.

```
/**Verifica se a1 appare come sottostringa contigua di a2.
 @param a1 Array1.
 @param a2 Array2.
 @return True/false.*/
public class Infix
{
 public static boolean infix(int[] a1, int[] a2)
 {
  for(int i=0;i<=a2.length-a1.length;i++)
  {
   if(a1[0]==a2[i])
     if(verifica(a1,a2,i)==true)
       return true;
  }
 return false;
}
```

```
/**Metodo ausiliario statico che trova se a1 appare in a2 a partire da una
posizione k.
 @param a1 Array1.
 @param a2 Array2.
 @return True/false.*/
 private static boolean verifica(int[] a1, int[] a2,int k)
 {
  int cont=-1;
  for(int i=k;i<k+a1.length;i++)
  {
   cont++;
   if(a1[cont]!=a2[i])
     return false;
  }
  return true;
 }
}


public class InfixTester
{
 public static void main(String[] args)
 {
  int[] a1=new int[4];
  a1[0]=6;
  a1[1]=2;
  a1[2]=1;
  a1[3]=0;
  int[] a2=new int[8];
  a2[0]=6;
  a2[1]=4;
  a2[2]=5;
  a2[3]=6;
  a2[4]=2;
  a2[5]=7;
  a2[6]=0;
  a2[7]=5;
  System.out.println(Infix.infix(a1,a2));
 }
}
```

Mix

Testo:

Sviluppare un metodo statico int[] mix(int[] a1, int[] a2) che accetta in ingresso due array di eguale lunghezza, e restituisce un array formato dal primo elemento di a1, seguito dal primo elemento di a2, seguito dal secondo elemento di a1, seguito dal secondo elemento di a2, e così via. Se i due array in ingresso non hanno eguale lunghezza, il metodo restituisce null.

Consigli:

Per risolvere questo esercizio è sufficiente un solo ciclo for. La parte difficile è l'inserimento dei valori in a3, per farlo si consiglia di moltiplicare l'indice per due (per gli indici pari di a3) e di moltiplicare per due e incrementare di una unità (per gli indici dispari di a3).

```
public class Mix
{
  public static int[] mix(int[] a1, int[] a2)
    {
     if(a1.length!=a2.length)
      return null;

     int[] a3= new int[a1.length*2];
     for(int i=0;i<a1.length;i++)
     {
      a3[i*2]=a1[i];
      a3[(i*2)+1]=a2[i];
     }
    return a3;
    }
}
```

```
import java.util.Arrays;
public class MixTester
{

 public static void main(String [] args)
      {
      int[] prova=new int[3];
    prova[0]=1;
    prova[1]=1;
    prova[2]=1;
      int[] prova2=new int[3];
    prova2[0]=2;
    prova2[1]=2;
    prova2[2]=2;


    System.out.println(Arrays.toString(Mix.mix(prova,prova2)));
    }
}
```

Disgiunte

Testo:

Sviluppare un metodo statico boolean disgiunte(String s1, String s2) che restituisce true se e solo se la stringa s1 non contiene alcun carattere della stringa s2, restituisce false altrimenti. Ad esempio, per le stringhe "fila" e "computer" il metodo restituisce true perché non vi è alcun carattere in comune.

Consigli:

Si consiglia l'utilizzo di due for innestati per confrontare i caratteri delle stringhe, appena se ne trova uno uguale si restituisce false.

```
public class Disgiunte
{
public static boolean disgiunte(String s1,String s2)
{
 if(s1.length()==0&&s2.length()==0)
   return false;
 for(int i=0;i<s1.length();i++)
 {
   for(int j=0;j<s2.length();j++)
   {
      if(s1.substring(i,i+1).equals(s2.substring(j,j+1)))
       return false;
   }
 }
 return true;
}
}
```

```
public class DisgiunteTester
{
  public static void main(String[] args )
    {
    String s1="computer";
    String s2="fila";

    System.out.println(Disgiunte.disgiunte(s1,s2));
    }
 }
```

Split

Testo:

Implementare un metodo public static int[] split(int[] unArray){...} che restituisce un array ottenuto spostando a sinistra tutti i numeri pari presenti in unArray, e spostando a destra tutti i numeri dispari, mantenendo l'ordine relativo degli elementi. Ad esempio, assegnato l'array [4, 13, 18, 7, 5, 8] il metodo dovrà restituire l'array [4, 18, 8, 13, 7, 5], dove l'ordine relativo degli elementi è rimasto invariato.

Consigli:

Si consiglia di dividere l'esercizio in due parti quindi ci saranno due for ma NON innestati(quindi si può usare sempre l'indice i). Prima si inseriscono tutti i numeri pari nel nuovo array (chiamato a), nella seconda parte si inseriscono i dispari. È stato creato un metodo stampa per vedere l'array a in uscita nel tester.

```
public class Split
{
public static int[] split(int[] unArray)
 {
 int conto=0;
 int[] a=new int[unArray.length];
  for(int i=0;i<unArray.length;i++)
  {
    if(unArray[i]%2==0)
    {
     a[conto]=unArray[i];
     conto++;
    }
  }

  for(int i=0;i<unArray.length;i++)
  {
    if(unArray[i]%2!=0)
    {
     a[conto]=unArray[i];
     conto++;
    }
  }
 return a;
 }
/**Stampa l'array (necessario per SplitTester).
@param array a.
@return array stampato sottoforma di stringa. */
    public static String stampa(int[] a)
    {
    String r="Risulta: ";
     for(int j=0;j<a.length;j++)
     {
      r=r+a[j];
     }
     return r;
    }
}
```

```
public class SplitTester
{
public static void main(String [] args)
{
int[] prova=new int[6];
    prova[0]=4;
    prova[1]=1;
    prova[2]=1;
    prova[3]=7;
    prova[4]=5;
    prova[5]=8;


    System.out.println(Split.stampa(Split.split(prova)));
}
}
```

Filter

Testo:

Implementare un metodo public static int[] filter(int[] unArray){...} che restituisce un array formato da tutti e soli i numeri dispari presenti in unArray, mantenendo il loro ordine relativo. Ad esempio, assegnato l'array [4, 13, 18, 7, 5, 8] il metodo dovr`a restituire l'array [13, 7, 5]. Utilizzare un metodo ausiliario per determinare la lunghezza del nuovo array che dovr`a essere restituito come risultato.

Consigli:

Si consiglia di utilizzare un metodo ausiliario per determinare la lunghezza del nuovo array che dovrà essere restituito come risultato. È stato creato un metodo stampa per vedere l'array a in uscita nel tester.

```
public class filter
 {

    public static int[] filter(int[] unArray)
    {
     int index=0;
     int[] filtrato =new int[lunghezza(unArray)];
     for(int i=0;i<unArray.length;i++)
     {
      if(unArray[i]%2!=0)
      {
      filtrato[index]=unArray[i];
      index++;
      }
     }
    return filtrato;
    }
  /**Metodo ausiliario che trova la unghezza del nuovo array.
   @param unArray array di partenza.
   @return lunghezza.*/
    private static int lunghezza(int[] unArray)
    {
     int cont=0;
     for(int i=0;i<unArray.length;i++)
     {
     if(unArray[i]%2!=0)
      cont++;
     }
    return cont;
    }
 /**Stampa l'array (necessario per SplitTester).
 @param array a.
 @return array stampato sottoforma di stringa. */
  public static String stampa(int[] a)
   {
    String s="";
    for(int i=0; i<a.length;i++)
    {
     s=s+" ,"+a[i];
    }
   return s;
   }
}
```

```
public class FilterTester
{
public static void main(String [] args)
    {
    int[] prova=new int[6];
    prova[0]=1;
    prova[1]=13;
    prova[2]=9;
    prova[3]=4;
    prova[4]=6;
    prova[5]=9;

    System.out.println(filter.stampa(filter.filter(prova)));

    }
}
```

INDICE